ROBOTICS

Technisches Referenzhandbuch

RAPID Instruktionen, Funktionen und Datentypen

Trace back information:
Workspace 21D version a2
Checked in 2021-11-30
Skribenta version 5.4.005

Technisches Referenzhandbuch
RAPID Instruktionen, Funktionen und Datentypen

RobotWare 6.13

Dokumentnr: 3HAC050917-003
Revision: P

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.
Spezifikationen können ohne vorherige Ankündigung geändert werden.

Die Informationen in diesem Handbuch können ohne vorherige Ankündigung geändert werden und stellen keine Verpflichtung von ABB dar. ABB übernimmt keinerlei Verantwortung für etwaige Fehler, die dieses Handbuch enthalten kann.

Wenn nicht ausdrücklich in vorliegendem Handbuch angegeben, gibt ABB für keine hierin enthaltenen Informationen Sachmängelhaftung oder Gewährleistung für Verluste, Personen- oder Sachschäden, Verwendbarkeit für einen bestimmten Zweck oder Ähnliches.

In keinem Fall kann ABB haftbar gemacht werden für Schäden oder Folgeschäden, die sich aus der Anwendung dieses Dokuments oder der darin beschriebenen Produkte ergeben.

Dieses Handbuch darf weder ganz noch teilweise ohne vorherige schriftliche Genehmigung von ABB vervielfältigt oder kopiert werden.

Zur späteren Verwendung aufbewahren.

Zusätzliche Kopien dieses Handbuchs können von ABB bezogen werden.

Übersetzung der Originalbetriebsanleitung.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.
Spezifikationen können ohne vorherige Ankündigung geändert werden.

Inhaltsverzeichnis

Überblick über dieses Handbuch 19

1 Instruktionen 25

1.1 AccSet - Verringern der Beschleunigung 25
1.2 ActEventBuffer – Aktivierung des Ereignispuffers 28
1.3 ActUnit - Aktivieren einer mechanischen Einheit 30
1.4 Add – Addiert einen numerischen Wert 32
1.5 AliasCamera - Definieren eines Kameragerät mit Aliasname 34
1.6 AliasIO - Definieren eines E/A-Signals mit Aliasname 37
1.7 AliasIOReset – Zurücksetzen eines E/A-Signals mit Aliasnamen 40
1.8 ":=" - Zuweisen eines Werts 42
1.9 BitClear – Löschen eines angegebenen Bits in Byte- oder dnum-Daten 44
1.10 BitSet – Setzen eines angegebenen Bits in Byte- oder dnum-Daten 47
1.11 BookErrNo - Vergeben einer RAPID-Systemfehlernummer 50
1.12 Break - Abbruch der Programmabarbeitung 52
1.13 CallByVar - Aufruf einer Prozedur durch eine Variable 53
1.14 CamFlush – Entfernt die erfassten Daten für die Kamera. 55
1.15 CamGetParameter – Ruft unterschiedlich benannte Kameraparameter ab. 56
1.16 CamGetResult – Ruft ein Kameraziel aus der Erfassung ab. 58
1.17 CamLoadJob – Laden Sie eine Kameratask auf die Kamera 60
1.18 CamReqImage – Senden Sie einen Befehl an die Kamera, ein Bild zu erfassen. 62
1.19 CamSetExposure – Legen Sie die spezifischen Daten der Kamera fest 64
1.20 CamSetParameter – Legen Sie unterschiedlich benannte Kameraparameter fest. 66
1.21 CamSetProgramMode – Weist die Kamera an, in den Programmmodus zu wechseln 68
1.22 CamSetRunMode – Weist die Kamera an, in den Abarbeitungsmodus zu wechseln 70
1.23 CamStartLoadJob – Starten Sie damit, eine Kameratask auf die Kamera zu laden 72
1.24 CamStartSetParameter – Start des Setzen-Vorgangs einer Parametereinstellung. 74
1.25 CamWaitLoadJob – Warten Sie, bis eine Kameratask geladen wird 77
1.26 CamWaitSetParameter – Warten, bis ein neuer Set-Vorgang bereit ist. 79
1.27 CancelLoad - Abbruch des Ladens eines Moduls 81
1.28 CapAPTrSetup - Einrichtung eines At-Point-Trackers 83
1.29 CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird. 86
1.30 CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird. 89
1.31 CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird. 92
1.32 CapC - Instruktion für kreisförmige CAP-Bewegung 95
1.33 CapCondSetDO - Setzt ein digitales Ausgangssignal bei TCP-Stopp 105
1.34 CapEquiDist Ereignis mit gleichem Abstand erstellen 107
1.35 CapL - Instruktion für lineare CAP-Bewegung 109
1.36 CapLATrSetup - Look-Ahead-Tracker einrichten 118
1.37 CapNoProcess - CAP ohne Prozess starten 123
1.38 CapRefresh - CAP-Daten aktualisieren 125
1.39 CAPSetStopMode - Stellen Sie den Stopp-Modus für Execution Error ein 127
1.40 CapWeaveSync - Signale und Ebenen für die Synchronisierung der Pendelbewegung einrichten. 128
1.41 CheckProgRef - Überprüfen von Programmverweisen 131
1.42 CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn 133
1.43 Clear - Löschen des Werts 139
1.44 ClearIOBuff - Löschen des Eingangspuffers eines seriellen Kanals 140
1.45 ClearPath - Löschen der aktuellen Bahn 142
1.46 ClearRawBytes - Löschen des Inhalts von rawbytes-Daten 146
1.47 ClkReset - Stellt eine Uhr zurück, die zur Zeitnahme verwendet wird 148
1.48 ClkStart - Starten einer Uhr, die zur Zeitnahme verwendet wird 149
1.49 ClkStop - Stoppen einer Uhr, die zur Zeitnahme verwendet wird 151
1.50 Schließen - Schließt eine Datei oder ein E/A-Gerät 152

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.51 CloseDir - Schließen eines Verzeichnisses 153
1.52 Comment - Kommentar 154
1.53 Compact IF - Wenn eine Bedingung erfüllt ist, dann... (eine Instruktion) 155
1.54 ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung 156
1.55 ConfL - Überwachen der Konfiguration während linearer Bewegung 158
1.56 CONNECT - Verbinden eines Interrupts mit einer Interrupt-Routine 161
1.57 ContactL – Lineare Kontaktbewegung 163
1.58 CopyFile - Kopieren einer Datei 169
1.59 CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten 171
1.60 CornerPathWarning – Zonenbahnwarnungen anzeigen oder verbergen 174
1.61 CorrClear - Entfernen aller Korrekturgeneratoren 176
1.62 CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator 177
1.63 CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator 182
1.64 CorrWrite - Schreiben an einen Korrekturgenerator 183
1.65 DeactEventBuffer – Deaktivierung des Ereignispuffers 185
1.66 DeactUnit - Deaktivieren einer mechanischen Einheit 187
1.67 Decr - Dekrementieren um 1 189
1.68 DropSensor - Ablegen des Objekts am Sensor 191
1.69 DropWObj - Ablegen des Werkobjekts am Conveyor 193
1.70 EOffsOff - Deaktivieren eines Offsets für zusätzliche Achsen 194
1.71 EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen 195
1.72 EOffsSet - Aktivieren eines Offsets für zusätzliche Achsen durch bekannte Werte 198
1.73 EraseModule - Löschen eines Moduls 200
1.74 ErrLog - Schreiben einer Fehlermeldung 202
1.75 ErrRaise - Schreiben einer Warnung und Aufrufen einer Fehlerbehandlung 206
1.76 ErrWrite - Schreiben einer Fehlermeldung 211
1.77 EXIT - Beenden der Programmabarbeitung 213
1.78 ExitCycle - Abbrechen des aktuellen und Starten des nächsten Zyklus 214
1.79 FitCircle – Passt einen Kreis an 3D-Punkte an 216
1.80 FOR - Vorgegebene Anzahl an Wiederholungen 220
1.81 FricIdInit - Initiierung der Reibungsidentifikation 222
1.82 FricIdEvaluate - Evaluierung der Reibungsidentifikation 223
1.83 FricIdSetFricLevels - Stellt Reibungslevel nach Reibungsidentifikation ein 226
1.84 GetDataVal - Abrufen des Werts eines Datenobjekts 228
1.85 GetGroupSignalInfo - Lesen der Informationen zu dem digitalen Gruppensignal 231
1.86 GetJoinData – Abrufen der gelenkspezifischen Daten 233
1.87 GetSysData - Abrufen von Systemdaten 235
1.88 GetTrapData - Abrufen von Interrupt-Daten für aktuellen Interrupt 238
1.89 GOTO - Sprung zu einer anderen Instruktion 240
1.90 GripLoad - Definieren der Nutzlast des Roboters 242
1.91 HollowWristReset - Hohlgelenk zurücksetzen 244
1.92 ICap - verbindet CAP-Ereignisse mit Trap-Routinen 246
1.93 IDelete - Abbrechen eines Interrupts 252
1.94 IDisable - Blockieren von Interrupts 253
1.95 IEnable - Freigeben von Interrupts 254
1.96 IError - Anfordern eines Interrupts bei Fehlern 255
1.97 IF - Wenn eine Bedingung erfüllt ist, dann...; sonst... 258
1.98 Incr - Inkrementieren um 1 260
1.99 IndAMove - Unabhängige Bewegung in absolute Position 262
1.100 IndCMove - Unabhängige kontinuierliche Bewegung 266
1.101 IndDMove - Unabhängige Delta-Position-Bewegung 270
1.102 IndReset - Unabhängiges Zurücksetzen 274
1.103 IndRMove - Unabhängige Bewegung an relative Position 279
1.104 InitSuperv - Zurücksetzen der gesamten CAP-Überwachung 284
1.105 InvertDO - Invertieren des Werts eines digitalen Ausgangssignals 285
1.106 IOBusStart - Start of I/O network 287
1.107 IOBusState - Abrufen des aktuellen Status des E/A-Netzwerks 288
1.108 IODisable – Deaktivieren eines E/A-Geräts 291
1.109 IOEnable – Aktivieren eines E/A-Geräts 294

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.110 IOEventMessage - E/A-Ereignismeldungen von Gerät ein-/ausschalten 297
1.111 IPathPos - Bei Pendelbewegug robtarget Mittellinie abrufen 299
1.112 IPers - Interrupt bei Wertänderung einer persistenten Variable 301
1.113 IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp 303
1.114 ISignalAI - Interrupts von analogem Eingangssignal 307
1.115 ISignalAO - Interrupts von analogem Ausgangssignal 316
1.116 ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal 320
1.117 ISignalDO - Interrupts von einem digitalen Ausgangssignal 324
1.118 ISignalGI - Anfordern von Interrupts von einer Gruppe digitaler Eingangssignale 328
1.119 ISignalGO - Anfordern von Interrupts von einer Gruppe digitaler Ausgangssignale 331
1.120 ISleep - Deaktivieren eines Interrupts 334
1.121 ITimer - Anfordern eines zeitgesteuerten Interrupts 336
1.122 IVarValue - Anfordern eines Interrupts mit variablem Wert 339
1.123 IWatch - Aktivieren eines Interrupts 343
1.124 Label - Zeilenname 345
1.125 Load - Laden eines Programmmoduls während der Abarbeitung 346
1.126 LoadId - Lastidentifikation der Werkzeug- oder Nutzlast 351
1.127 MakeDir - Erstellen eines neuen Verzeichnisses 357
1.128 ManLoadIdProc - Lastidentifikation des Manipulators IRBP 359
1.129 MatrixSolve - Lösen eines linearen Gleichungssystems 363
1.130 MatrixSolveQR - Berechnen einer QR-Faktorisierung 366
1.131 MatrixSVD - Berechnen einer Einzelwertzerlegung 368
1.132 MechUnitLoad - Definieren einer Nutzlast für eine mechanische Einheit 372
1.133 MotionProcessModeSet - Einstellen des Bewegungsprozess-Modus 377
1.134 MotionSup - Deaktivieren/Aktivieren der Bewegungsüberwachung 379
1.135 MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition 382
1.136 MoveC - Kreisförmige Bewegung des Roboters 389
1.137 MoveCAO - Für eine kreisförmige Bewegung des Roboters und der Festlegung eines analogen Ausgangs in der Ecke 398
1.138 MoveCDO – Der Roboter wird kreisförmig bewegt und es wird ein digitaler Ausgang in der Zonenbahn gesetzt 403
1.139 MoveCGO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des Gruppenausgangssignals in der Ecke 408
1.140 MoveCSync - Kreisförmiges Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur 413
1.141 MoveExtJ - Bewegen einer oder mehrerer mechanischer Einheiten ohne TCP 418
1.142 MoveJ - Achsenbewegung des Roboters 422
1.143 MoveJAO - Für eine Achsenbewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke 428
1.144 MoveJDO - Achsenbewegung des Roboters und Setzen eines digitalen Ausgangs in der Zonenbahn 433
1.145 MoveJGO - Für eine Achsenbewegung des Roboters und Einstellen eines Gruppenausgangs in der Ecke 438
1.146 MoveJSync - Bewegen des Roboters durch Achsbewegung und Abarbeiten einer RAPID-Prozedur 443
1.147 MoveL - Lineare Bewegung des Roboters 448
1.148 MoveLAO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke 455
1.149 MoveLDO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke 460
1.150 MoveLGO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke 465
1.151 MoveLSync - Lineares Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur 470
1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn 475
1.153 MToolRotCalib - Kalibrierung der Rotation eines Werkzeugs in Bewegung 486
1.154 MToolTCPCalib - Kalibrierung des TCP eines Werkzeugs in Bewegung 489
1.155 Open - Öffnet eine Datei oder ein E/A-Gerät 492
1.156 OpenDir - Öffnen eines Verzeichnisses 497
1.157 PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten 499

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.158 PackRawBytes - Packen von Daten in rawbytes-Daten 502
1.159 PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn 507
1.160 PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück 511
1.161 PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht 513
1.162 PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht 515
1.163 PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen 517
1.164 PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen 525
1.165 PathRecStart - Starten der Bahnaufzeichnung 528
1.166 PathRecStop - Stoppen der Bahnaufzeichnung 531
1.167 PathResol - Override der Bahnauflösung 534
1.168 PDispOff - Deaktivieren der Programmverschiebung 536
1.169 PDispOn - Aktivieren der Programmverschiebung 537
1.170 PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem 542
1.171 ProcCall - Aufrufen einer neuen Prozedur 545
1.172 ProcerrRecovery - Generieren und Beheben von Bewegungsfehlern 547
1.173 PrxActivAndStoreRecord - Aktiviert und speichert die aufgezeichneten Profildaten 553
1.174 PrxActivRecord - Aktiviert die aufgezeichneten Profildaten 555
1.175 PrxDbgStoreRecord - Speichert und prüft die aufgezeichneten Profildaten 557
1.176 PrxDeactRecord - Deaktiviert eine Aufzeichnung 558
1.177 PrxResetPos - Setzt die Nullposition des Sensors zurück 559
1.178 PrxResetRecords - Deaktiviert alle Aufzeichnungen und setzt diese zurück 560
1.179 PrxSetPosOffset - Legt eine Referenzposition für den Sensor fest 561
1.180 PrxSetRecordSampleTime - Legt die Abtastzeit für die Aufzeichnung eines Profils fest ... 562
1.181 PrxSetSyncalarm - Verhalten bei Sync-Alarmen festlegen 563
1.182 PrxStartRecord - Aufzeichnung eines neuen Profils 565
1.183 PrxStopRecord - Aufzeichnung eines Profils anhalten 567
1.184 PrxStoreRecord - Speichert die aufgezeichneten Profildaten 568
1.185 PrxUseFileRecord - Verwendet die aufgezeichneten Profildaten 570
1.186 PulseDO - Erzeugen eines Impulses auf einem digitalen Ausgangssignal 572
1.187 RAISE - Ruft die Fehlerbearbeitung auf 575
1.188 RaiseToUser - Weitergeben eines Fehlers an die Benutzerebene 578
1.189 ReadAnyBin – Lesen von Daten aus einem binären E/A-Gerät oder einer Binärdatei 581
1.190 ReadBlock - Lesen eines Datenblocks von einem Gerät 584
1.191 ReadCfgData - Lesen von Attributen eines Systemparameters 586
1.192 ReadErrData - Abrufen von Informationen über einen Fehler 590
1.193 ReadRawBytes - Lesen der rawbytes-Daten 593
1.194 ReadVarArr - Mehrere Variablen von einem Sensorgerät ablesen 596
1.195 RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen 599
1.196 RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung 601
1.197 RemoveDir - Löschen eines Verzeichnisses 603
1.198 RemoveFile - Löschen einer Datei 605
1.199 RemoveSuperv Bedingung für ein Signal entfernen 607
1.200 RenameFile - Umbenennen einer Datei 609
1.201 Reset - Setzt ein digitales Ausgangssignal zurück 611
1.202 ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse 613
1.203 ResetAxisMoveTime - Zurücksetzen des Bewegungszeitzählers der Achse 615
1.204 ResetPPMoved - Zurücksetzen des Status für den im Einrichtbetrieb verschobenen Programmzeiger 617
1.205 ResetRetryCount - Zurücksetzen der Anzahl an Wiederholungen 618
1.206 ResetTorqueMargin - Die geringste Drehmomentspanne zurückstellen 619
1.207 RestoPath - Wiederherstellen der Bahn nach einem Interrupt 620
1.208 RETRY - Wiederaufnahme der Abarbeitung nach einem Fehler 622
1.209 RETURN - Beenden der Abarbeitung einer Routine 624
1.210 Rewind - Zurücksetzen des Dateizeigers 626
1.211 RMQEmptyQueue - Leere RAPID Message Queue 627
1.212 RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens 629
1.213 RMQGetMessage - Abrufen einer RMQ-Meldung 631

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.214 RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung 634
1.215 RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung 637
1.216 RMQReadWait – Rückgabe der Meldung von der RMQ 640
1.217 RMQSendMessage - Senden einer RMQ-Datenmeldung 643
1.218 RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort 647
1.219 SafetyControllerSyncRequest - Initiierung des Hardware-Synchronisationsverfahrens 652
1.220 Save - Speichern eines Programmmoduls 653
1.221 SaveCfgData - Speichert Systemparameter in einer Datei 656
1.222 SCWrite - Senden von Variablendaten an eine Client-Anwendung 658
1.223 SearchC - Kreisförmige Suchbewegung des Roboters 661
1.224 SearchExtJ - Suchen mit einer oder mehreren mechanischer Einheiten ohne TCP 672
1.225 SearchL - Lineare Suchbewegung des Roboters 681
1.226 SenDevice - Verbinden mit einer Sensoreinheit 693
1.227 Set - Setzen eines digitalen Ausgangssignals 695
1.228 SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge 697
1.229 SetAO - Ändern des Werts eines analogen Ausgangssignals 700
1.230 SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird 702
1.231 SetDataVal - Setzen des Werts eines Datenobjekts 707
1.232 SetDO - Ändern des Werts eines digitalen Ausgangssignals 710
1.233 SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen 713
1.234 SetLeadThrough - Durchführung aktivieren und deaktivieren 717
1.235 SetSysData - Setzen von Systemdaten 720
1.236 SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung 722
1.237 SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP 725
1.238 SiConnect - Sensorschnittstelle verbinden 728
1.239 SiClose - Sensorschnittstelle schließen 731
1.240 SiGetCyclic - Sensorschnittstelle zyklisch abrufen 733
1.241 SimCollision - Simulation einer Kollision 735
1.242 SingArea - Definieren der Interpolation in der Nähe singulärer Punkte 736
1.243 SiSetCyclic - Sensorschnittstelle zyklisch einstellen 739
1.244 SkipWarn - Überspringen der neuesten Warnmeldung 741
1.245 SocketAccept - Akzeptieren einer eingehenden Verbindung 742
1.246 SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port 746
1.247 SocketClose - Schließen eines Sockets 749
1.248 SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer 751
1.249 SocketCreate - Erstellen eines neuen Sockets 754
1.250 SocketListen - Prüfen auf eingehende Verbindungen 757
1.251 SocketReceive - Empfangen von Daten vom Remote-Computer 759
1.252 SocketReceiveFrom - Daten vom Remote-Computer empfangen 764
1.253 SocketSend - Senden von Daten an Remote-Computer 769
1.254 SocketSendTo - Daten an Remote-Computer senden 774
1.255 SoftAct - Aktivieren von Softservo 779
1.256 SoftDeact - Deaktivieren von Softservo 781
1.257 SoftElbow – Sorgt für Flexibilität des Ellenbogenstücks beim Einwirken von externen Kräften. 782
1.258 SpeedLimAxis – Einstellen der Geschwindigkeitsbegrenzung für eine Achse 784
1.259 SpeedLimCheckPoint – Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte 789
1.260 SpeedRefresh – Geschwindigkeitsübersteuerung für laufende Bewegung aktualisieren .. 795
1.261 SpyStart - Starten der Aufzeichnung von Abarbeitungsdaten 798
1.262 SpyStop - Stoppen der Aufzeichnung von Abarbeitungsdaten 801
1.263 StartLoad – Laden eines Programmmoduls während der Abarbeitung 802
1.264 StartMove - Erneutes Starten der Roboterbewegung 806
1.265 StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung 809
1.266 STCalib - Kalibrieren eines Servowerkzeugs 812
1.267 STClose - Schließen eines Servowerkzeugs 817
1.268 StepBwdPath - Rückwärtsbewegung um einen Schritt auf der Bahn 821
1.269 STIndGun - Versetzen des Servowerkzeug in den Modus „Unabhängig“. 823
1.270 STIndGun - Zurücksetzen des Werkzeugs aus dem Modus „Unabhängig“. 825
1.271 SToolRotCalib - Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug 826

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.272 SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs 829
1.273 Stop - Stoppt die Programmabarbeitung 833
1.274 STOpen - Öffnen eines Servowerkzeugs 836
1.275 StopMove - Stoppen der Roboterbewegung 839
1.276 StopMoveReset - Zurücksetzen des Systemstatus für das Stoppen der Bewegung 843
1.277 StorePath - Speichern der Bahn bei einem Interrupt 846
1.278 STTune - Abstimmung des Servowerkzeugs 849
1.279 STTuneReset - Zurücksetzen der Servowerkzeugabstimmung 853
1.280 SupSyncSensorOff - Stopp der synchronisierten Sensorüberwachung 854
1.281 SupSyncSensorOn - Start der synchronisierten Sensorüberwachung 855
1.282 SyncMoveOff - Beenden von koordinierten Synchronbewegungen 857
1.283 SyncMoveOn - Beginn von koordinierten Synchronbewegungen 863
1.284 SyncMoveResume - Festlegen von synchronisierten koordinierten Bewegungen 869
1.285 SyncMoveSuspend - Festlegen von unabhängigen teilkoordinierten Bewegungen 871
1.286 SyncMoveUndo - Festlegen von unabhängigen Bewegungen 873
1.287 SyncToSensor - Synchr. mit Sensor 875
1.288 SystemStopAction - Anhalten des Robotersystems 877
1.289 TEST - Abhängig vom Wert eines Ausdrucks 879
1.290 TestSignDefine - Definieren eines Prüfsignals 881
1.291 TestSignReset - Zurücksetzen aller Prüfsignaldefinitionen 883
1.292 TextTabInstall - Installieren einer Texttabelle 884
1.293 TPErase - Löschen des Texts am FlexPendant 886
1.294 TPReadDnum - Lesen eines numerischen Werts am FlexPendant 887
1.295 TPReadFK - Lesen der Funktionstasten 891
1.296 TPReadNum - Lesen eines numerischen Werts am FlexPendant 896
1.297 TPShow - Wechsel des Fensters am FlexPendant 900
1.298 TPWrite - Schreiben am FlexPendant 901
1.299 TriggC - Kreisförmige Roboterbewegung mit Ereignissen 904
1.300 TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position 913
1.301 TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable 919
1.302 TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen 921
1.303 TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn 923
1.304 TriggInt - Definieren eines positionsabhängigen Interrupts 930
1.305 TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt 935
1.306 TriggJ - Achsenweise Roboterbewegungen mit Ereignissen 942
1.307 TriggL - Lineare Roboterbewegungen mit Ereignissen 950
1.308 TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen 959
1.309 TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen 967
1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn 975
1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung 983
1.312 TriggStopProc - Generieren von Neustartdaten für Trigg-Signale bei einem Stopp 994
1.313 TryInt - Überprüfen, ob ein Datenobjekt eine gültige Ganzzahl ist 1000
1.314 TRYNEXT - Überspringen einer fehlerverursachenden Instruktion 1002
1.315 TuneReset - Zurücksetzen der Servo-Abstimmung 1003
1.316 TuneServo - Abstimmen von Servos 1004
1.317 UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ 1011
1.318 UIMsgWrite - Benutzer-Meldungsfenster, Typ nicht-wartend 1020
1.319 UIMsgWriteAbort - Benutzer-Meldungsfenster, Typ nicht-wartend, stornieren 1025
1.320 UIShow - Anzeigen der Benutzeroberfläche 1026
1.321 UnLoad - Entladen eines Programmmoduls während der Abarbeitung 1030
1.322 UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten 1033
1.323 VelSet - Ändern der programmierten Geschwindigkeit 1038
1.324 WaitAI - Warten, bis ein analoger Eingangssignalwert gesetzt wird 1040
1.325 WaitAO - Warten, bis ein analoger Ausgangssignalwert gesetzt wird 1046
1.326 WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist 1052

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.327 WaitDO - Warten, bis ein digitales Ausgangssignal gesetzt ist 1058
1.328 WaitGI - Warten, bis eine Gruppe digitaler Eingangssignale gesetzt wird 1064
1.329 WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird 1071
1.330 WaitLoad – Das geladene Modul mit der Task verbinden 1078
1.331 WaitRob – Bis zum Stopppunkt oder Stillstand warten 1083
1.332 WaitSensor - Auf die Sensorverbindung warten 1085
1.333 WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten 1089
1.334 WaitTestAndSet – Warten, bis Variable FALSE wird, dann einstellen 1093
1.335 WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist 1096
1.336 WaitUntil – Warten, bis eine Bedingung erfüllt ist 1098
1.337 WaitWObj - Warten auf Werkobjekt am Förderer 1106
1.338 WarmStart – Die Steuerung neu starten 1109
1.339 WHILE – So lange wiederholen bis... 1110
1.340 WorldAccLim - Steuern der Beschleunigung im Welt-Koordinatensystem 1112
1.341 Write – In eine zeichenbasierte Datei oder ein E/A-Gerät schreiben 1114
1.342 WriteAnyBin - Schreibt Daten in eine zeichenorientierte Datei oder ein zeichenorientiertes E/A-Gerät 1117
1.343 WriteBin - Schreibt in ein binäres E/A-Gerät 1119
1.344 WriteBlock – Datenblock auf Gerät schreiben 1121
1.345 WriteCfgData - Schreibt ein Attribut eines Systemparameters 1123
1.346 WriteRawBytes - Schreiben von rawbytes-Daten 1127
1.347 WriteStrBin - Schreibt einer Zeichenfolge auf ein binäres E/A-Gerät 1130
1.348 WriteVar – Variable schreiben 1132
1.349 WriteVarArr - Schreiben mehrerer Variablen auf ein Sensorgerät 1135
1.350 WZBoxDef - Definieren einer quaderförmigen Weltzone 1137
1.351 WZCylDef - Definieren einer zylinderförmigen Weltzone 1139
1.352 WZDisable - Deaktivieren der temporären Weltzonenüberwachung 1142
1.353 WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs 1144
1.354 WZEnable - Aktivieren einer temporären Weltzonenüberwachung 1149
1.355 WZFree - Löschen der temporären Weltzonenüberwachung 1151
1.356 WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen 1153
1.357 WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung 1157
1.358 WZLimSup - Aktivieren der Überwachung der Weltzonengrenze 1161
1.359 WZSphDef - Definieren einer kugelförmigen Weltzone 1164

2 Funktionen 1167

2.1 Abs - Berechnen des Absolutwerts 1167
2.2 AbsDnum - Ruft den absoluten Wert eines dnum ab 1169
2.3 ACos - Berechnen des Arcuscosinus 1171
2.4 ACosDnum - Berechnen des Arcuscosinus 1172
2.5 AInput - Lesen des Wertes eines analogen Eingangssignals 1173
2.6 AND – Bewertet einen logischen Wert 1175
2.7 AOutput - Lesen des Werts eines analogen Ausgangssignals 1177
2.8 ArgName - Abrufen des Argumentnamens 1179
2.9 ASin - Berechnen den Arcussinus 1183
2.10 ASinDnum - Berechnen den Arcussinus 1184
2.11 ATan - Berechnen des Arcustangens 1185
2.12 ATanDnum - Berechnen des Arcustangens 1186
2.13 ATan2 - Berechnen des Arcustangens2 1187
2.14 ATan2Dnum - Berechnen des Arcustangens2 1188
2.15 BitAnd - Logische bitweise AND-Operation mit Byte-Daten 1189
2.16 BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten 1191
2.17 BitCheck - Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist 1193
2.18 BitCheckDnum – Prüfen, ob ein angegebenes Bit in den dnum-Daten gesetzt ist 1195
2.19 BitLSh - Logische bitweise LEFT SHIFT-Operation mit Byte-Daten 1197
2.20 BitLShDnum – Logische bitweise LEFT SHIFT-Operation mit dnum-Daten 1199
2.21 BitNeg - Logische bitweise NEGATION-Operation mit Byte-Daten 1202
2.22 BitNegDnum – Logische bitweise NEGATION-Operation mit dnum-Daten 1204
2.23 BitOr - Logische bitweise OR-Operation mit Byte-Daten 1206

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.24 BitOrDnum – Logische bitweise OR-Operation mit dnum-Daten 1208
2.25 BitRSh - Logische bitweise RIGHT SHIFT-Operation mit Byte-Daten 1210
2.26 BitRShDnum – Logische bitweise RIGHT SHIFT-Operation mit dnum-Daten 1212
2.27 BitXOr - Logische bitweise XOR-Operation mit Byte-Daten 1214
2.28 BitXOrDnum – Logische bitweise XOR-Operation mit dnum-Daten 1216
2.29 ByteToStr - Konvertieren von byte-Daten in string-Daten 1218
2.30 CalcJointT - Berechnen der Achsenwinkel vom Roboterziel 1220
2.31 CalcRobT - Berechnen des Roboterziels aus dem Achsenziel 1225
2.32 CalcRotAxFrameZ – Berechnen eines Rotationsachsenrahmens 1228
2.33 CalcRotAxisFrame – Berechnen eines Rotationsachsenrahmens 1233
2.34 CamGetExposure – Abrufen der kameraspezifischen Daten 1237
2.35 CamGetLoadedJob – Abrufen des Namens der geladenen Kameratask 1239
2.36 CamGetMode - Aktuellen Modus der Kamera abrufen 1241
2.37 CamGetName – Abrufen des Namens der verwendeten Kamera 1242
2.38 CamNumberOfResults – Abrufen der Anzahl der verfügbaren Ergebnisse 1243
2.39 CapGetFailSigs - Ausgefallene E/A-Signale abrufen 1245
2.40 CDate - Lesen des aktuellen Datums als Zeichenfolge 1247
2.41 CJointT - Lesen der aktuellen Achsenwinkel 1248
2.42 ClkRead - Ablesen einer Uhr, die zur Zeitnahme verwendet wird 1250
2.43 CorrRead - Lesen der aktuellen Gesamt-Offsets 1252
2.44 Cos - Berechnen des Cosinus 1253
2.45 CosDnum - Berechnen des Cosinus 1254
2.46 CPos - Lesen der aktuellen Positionsdaten (pos) 1255
2.47 CRobT - Lesen der aktuellen Positionsdaten (Roboterziel) 1257
2.48 CrossProd - Kreuzprodukte von zwei Positionsvektoren berechnen. 1261
2.49 CSpeedOverride - Lesen der aktuellen Overridegeschwindigkeit 1264
2.50 CTime - Lesen der aktuellen Uhrzeit als Zeichenfolge 1266
2.51 CTool - Lesen der aktuellen Werkzeugdaten 1267
2.52 CWObj - Lesen der aktuellen Werkobjektdaten 1269
2.53 DecToHex - Konvertieren von Dezimal- in Hexadezimalformat 1271
2.54 DefAccFrame - Definieren eines präzisen Koordinatensystems 1272
2.55 DefDFrame - Definieren eines Verschiebungs-Koordinatensystems 1275
2.56 DefFrame - Definieren eines Koordinatensystems 1278
2.57 Dim - Abrufen der Dimension eines Datenfelds 1282
2.58 DInput - Lesen des Wertes eines digitalen Eingangssignals 1284
2.59 Distance – Abstand zwischen zwei Punkten 1286
2.60 DIV - berechnet eine Ganzzahldivision 1288
2.61 DnumToNum – Konvertieren von dnum in num 1289
2.62 DnumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge 1291
2.63 DotProd - Berechnen des Skalarprodukts von zwei pos-Vektoren 1293
2.64 DOutput - Lesen des aktuellen Werts von einem digitalen Ausgangssignal 1295
2.65 EulerZYX - Berechnen von Eulerschen Winkeln aus "orient" 1297
2.66 EventType – Abrufen des aktuellen Ereignistyps in einer beliebigen Ereignisroutine 1299
2.67 ExecHandler - Abrufen des Typs der Abarbeitungsbehandlung 1301
2.68 ExecLevel – Abarbeitungsebene abrufen 1302
2.69 Exp - Berechnen des Exponentialwerts 1303
2.70 FileSize - Abrufen der Größe einer Datei 1304
2.71 FileTimeDnum - Abrufen von Zeitinformationen über eine Datei 1307
2.72 FSSize - Abrufen der Größe eines Dateisystems 1310
2.73 GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse 1313
2.74 GetAxisMoveTime - Abrufen des Bewegungszeitzählers der Achse 1315
2.75 GetMaxNumberOfCyclicBool - Maximale Anzahl der Cyclic bool-Bedingungen abrufen 1317
2.76 GetMecUnitName – Abrufen des Namens der mechanischen Einheit 1318
2.77 GetModalPayLoadMode – Berechnen des ModalPayLoadMode-Werts 1319
2.78 GetMotorTorque – Liest das aktuelle Motordrehmoment. 1320
2.79 GetNextCyclicBool - Die Namen aller Cyclic bools abrufen 1323
2.80 GetNextMechUnit – Abrufen des Namens und der Daten für mechanische Einheiten 1325
2.81 GetNextSym - Abrufen des nächsten passenden Symbols 1328
2.82 GetNumberOfCyclicBool - Anzahl der Cyclic bool-Bedingungen abrufen 1330

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.83 GetServiceInfo - Abrufen von Informationen über das System 1331
2.84 GetSignalOrigin - Abruf von Informationen zur Herkunft eines E/A-Signals. 1333
2.85 GetSysInfo - Abrufen von Informationen über das System 1335
2.86 GetTaskName - Abrufen des Namens und der Nummer der aktuellen Task 1338
2.87 GetTime - Lesen der aktuellen Zeit als numerischer Wert 1340
2.88 GetTorqueMargin - Liest die geringste Drehmomentspanne ab 1342
2.89 GetTSPStatus – Rufen Sie den aktuellen Status des Taskauswahlfeldes ab 1344
2.90 GetUASUserName - Abfrage des Benutzernamens für Benutzer 1346
2.91 GInput - Liest den Wert des Gruppeneingangssignals. 1348
2.92 GInputDnum - Liest den Wert des Gruppeneingangssignals. 1350
2.93 GOutput - Lesen des Werts von einer Gruppe digitaler Ausgangssignale 1353
2.94 GOutputDnum – Lesen des Werts der Gruppe Ausgangssignale 1355
2.95 HexToDec - Konvertieren von Hexadezimal- in Dezimalformat 1358
2.96 IndInpos - Unabhängige Achse in Positionsstatus 1359
2.97 IndSpeed - Unabhängiger Geschwindigkeitsstatus 1361
2.98 IOUnitState - Abrufen des aktuellen Status des E/A-Geräts 1363
2.99 IsBrakeCheckActive – Überprüfen Sie, ob die Bremsprüfung ausgeführt wird 1366
2.100 IsCollFree - Überprüft die Position auf eine mögliche Kollision 1367
2.101 IsCyclicBool - Prüft, ob eine persistente Variable eine Cyclic bool ist 1369
2.102 IsFile - Prüfen des Dateityps 1372
2.103 IsLeadThrough - Status der Durchführung prüfen 1376
2.104 IsMechUnitActive - Prüfen, ob mechanische Einheit aktiv ist 1378
2.105 IsPers - Ist eine Persistente 1379
2.106 IsStopMoveAct - Sind Flags zum Stoppen der Bewegung aktiv? 1381
2.107 IsStopStateEvent – Prüfen, ob Programmzeiger bewegt wurde 1383
2.108 IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus 1385
2.109 IsSysId - Testen der Systemidentität 1387
2.110 IsVar - Prüfen, ob Datenobjekt eine Variable ist 1388
2.111 Max - Gibt den größten von zwei Werten zurück. 1390
2.112 MaxExtLinearSpeed - Maximale zusätzliche Achsengeschwindigkeit 1391
2.113 MaxExtReorientSpeed - Maximale Zusatzachsen-Rotationsgeschwindigkeit 1392
2.114 MaxRobReorientSpeed - Maximale Umorientierungsgeschwindigkeit des Roboters 1393
2.115 MaxRobSpeed - Höchstgeschwindigkeit des Roboters 1394
2.116 Min - Gibt den kleinsten von zwei Werten zurück. 1395
2.117 MirPos - Spiegeln einer Position 1396
2.118 MOD - berechnet einen Ganzzahlmodulo 1398
2.119 ModExist - Überprüfen, ob ein Programmmodul vorhanden ist. 1399
2.120 ModTimeDnum - Abrufen des Dateiänderungszeitpunkts für das geladene Modul 1400
2.121 MotionPlannerNo - Abrufen der verbundenen Bewegungsplanungsnummer 1402
2.122 NonMotionMode - Lesen des Nichtbewegungs-Abarbeitungsmodus 1404
2.123 NOT – Kehrt einen logischen Wert um 1406
2.124 NOrient - Normalisieren der Orientierung 1407
2.125 NumToDnum – Konvertieren von num in dnum 1409
2.126 NumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge 1410
2.127 Offs - Verschieben einer Roboterposition 1412
2.128 OpMode - Lesen des Betriebsmodus 1414
2.129 OR – Bewertet einen logischen Wert 1415
2.130 OrientZYX - Erzeugt eine Orientierung aus Eulerschen Winkeln. 1417
2.131 ORobT - Entfernen einer Programmverschiebung von einer Position 1419
2.132 ParIdPosValid - Gültige Roboterposition für Parameteridentifikation 1421
2.133 ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation 1424
2.134 PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus 1427
2.135 PathLevel - Abrufen der aktuellen Bahnebene 1429
2.136 PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde 1431
2.137 PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde 1434
2.138 PFRestart - Prüfen der unterbrochenen Bahn nach Stromausfall 1438
2.139 PoseInv - Invertieren einer Koordinatenverschiebung 1439
2.140 PoseMult - Multiplizieren von Koordinatenverschiebungen 1441
2.141 PoseVect - Anwenden einer Transformation auf einen Vektor 1443

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.142 Pow - Berechnen eines Potenzwerts 1445
2.143 PowDnum - Berechnen eines Potenzwerts 1446
2.144 PPMovedInManMode - Überprüfen, ob der Programmzeiger im Einrichtbetrieb verschoben wurde 1447
2.145 Present - Prüft, ob ein optionaler Parameter verwendet wird 1448
2.146 ProgMemFree - Abrufen der Größe des freien Programmspeichers 1450
2.147 PrxGetMaxRecordpos - Ruft die maximale Sensorposition ab 1451
2.148 Rand – Erzeugen Sie eine Zufallszahl 1452
2.149 RawBytesLen - Abrufen der Länge von rawbytes-Daten 1454
2.150 ReadBin - Liest ein Bytes aus einer Datei oder einem E/A-Gerät 1456
2.151 ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis 1459
2.152 ReadMotor - Liest die aktuellen Motorwinkel. 1462
2.153 ReadNum - Liest eine Zahl aus einer Datei oder einem E/A-Gerät 1464
2.154 ReadStr - Liest eine Zeichenfolge aus einer Datei oder einem E/A-Gerät 1467
2.155 ReadStrBin - Liest Zeichenfolge von binären E/A-Gerät oder einer Datei 1472
2.156 ReadVar - Lesen einer Variablen von einem Gerät 1474
2.157 RelTool - Verschiebung relativ zum Werkzeug 1476
2.158 RemainingRetries Restliche Anzahl an Wiederholungen 1478
2.159 RMQGetSlotName - Abrufen des Namens eines RMQ-Clients 1479
2.160 RobName - Abrufen des TCP-Roboternamens 1481
2.161 RobOS - Prüfen, ob die Abarbeitung auf RC oder VC erfolgt 1483
2.162 Round - Rundet einen numerischen Wert 1484
2.163 RoundDnum - Rundet einen numerischen Wert 1486
2.164 RunMode - Lesen des Abarbeitungsmodus 1488
2.165 SafetyControllerGetChecksum - Abrufen der Prüfsumme der Benutzerkonfigurationsdatei 1490
2.166 SafetyControllerGetOpModePinCode - Pin-Code der Betriebsart abrufen 1491
2.167 SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung 1492
2.168 SafetyControllerGetUserChecksum - Prüfsumme für geschützte Parameter abrufen 1493
2.169 Sin - Berechnet den Sinuswert. 1494
2.170 SinDnum - Berechnet den Sinuswert 1495
2.171 SocketGetStatus - Abrufen des aktuellen Socket-Status 1496
2.172 SocketPeek – Überprüfung eines Sockets auf vorhandene Daten hin 1499
2.173 Sqrt - Berechnen der Quadratwurzel 1502
2.174 SqrtDnum - Berechnen der Quadratwurzel 1503
2.175 STCalcForce - Berechnen der Elektrodenkraft für ein Servowerkzeug 1504
2.176 STCalcTorque - Berechnen des Motordrehmoments für ein Servowerkzeug 1506
2.177 STIsCalib - Überprüfen, ob ein Servowerkzeug kalibriert ist 1508
2.178 STIsClosed - Prüfen, ob ein Servowerkzeug geschlossen ist 1510
2.179 STIsIndGun - Überprüfen, ob sich ein Servowerkzeug im unabhängigen Modus befindet 1512
2.180 STIsOpen - Prüfen, ob ein Servowerkzeug geöffnet ist 1513
2.181 StrDigCalc - Arithmetische Operationen mit dem Datentyp stringdig 1515
2.182 StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten 1518
2.183 StrFind - Suchen nach einem Zeichen in einer Zeichenfolge 1520
2.184 StrFormat - Formatzeichenfolge. 1522
2.185 StrLen - Abrufen der Zeichenfolgenlänge 1524
2.186 StrMap - Zuordnen einer Zeichenfolge 1525
2.187 StrMatch - Suchen nach einem Muster in einer Zeichenfolge 1527
2.188 StrMemb - Prüfen, ob ein Zeichen einer Zeichenmenge angehört 1529
2.189 StrOrder - Prüfen, ob Zeichenfolgen geordnet sind 1531
2.190 StrPart - Suchen eines Teils der Zeichenfolge 1533
2.191 StrToByte - Konvertieren von string-Daten in Byte-Daten 1535
2.192 StrToVal - Konvertieren einer Zeichenfolge in einen Wert 1537
2.193 Tan - Berechnet den Tangens. 1539
2.194 TanDnum - Berechnen des Tangens 1540
2.195 TaskRunMec - Überprüfen, ob die Task eine mechanische Einheit steuert 1541
2.196 TaskRunRob - Überprüfen, ob die Task einen Roboter steuert 1542
2.197 TasksInSync – Gibt die Anzahl der synchronisierten Tasks zurück 1543

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.198 TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist 1545
2.199 TaskIsExecuting – Überprüfen Sie, ob die Task ausgeführt wird 1547
2.200 TestAndSet - Prüfen und ggf. Setzen einer nicht gesetzten Variablen 1549
2.201 TestDI - Prüfen, ob ein digitaler Eingang gesetzt ist 1552
2.202 TestSignRead - Lesen eines Prüfsignalwerts 1554
2.203 TextGet - Abrufen von Text aus Systemtexttabellen 1556
2.204 TextTabFreeToUse - Prüfen, ob Texttabelle frei ist 1559
2.205 TextTabGet - Abrufen der Texttabellennummer 1561
2.206 TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist 1563
2.207 Trunc - Schneidet einen numerischen Wert ab 1565
2.208 TruncDnum - Schneidet einen numerischen Wert ab 1567
2.209 Type - Abrufen des Datentypnamens für eine Variable. 1569
2.210 UIAlphaEntry - Benutzer-Alpha-Eintrag 1571
2.211 UIClientExist - Vorhandensein eines Benutzer-Clients 1578
2.212 UIDnumEntry - Eintrag eines numerischen Werts durch den Benutzer 1580
2.213 UIDnumTune - Benutzer-Zahlensynchronisierung 1588
2.214 UIListView - Anzeigen der Benutzerliste 1596
2.215 UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ 1605
2.216 UINumEntry - Eintrag eines numerischen Werts durch den Benutzer 1614
2.217 UINumTune - Benutzer-Zahlensynchronisierung 1621
2.218 ValidIO - Gültiges E/A-Signal, auf das zugegriffen werden soll 1628
2.219 ValToStr - Konvertieren eines Werts in eine Zeichenfolge 1630
2.220 Vectmagn - Berechnen der Länge eines pos-Vektors 1632
2.221 XOR – Bewertet einen logischen Wert 1634

3 Datentypen 1635

3.1 aiotrigg - Bedingung für analogen E/A-Trigger 1635
3.2 ALIAS – Zuweisen eines Alias-Datentyps 1637
3.3 bool – Logische Werte 1638
3.4 btnres - Ergebnisdaten für Drucktaste 1639
3.5 busstate - Status des E/A-Netzwerks 1641
3.6 buttondata – Drucktastendaten 1642
3.7 byte – Ganzzahlige Werte 0-255 1644
3.8 cameradev – Kameragerät 1645
3.9 camerastatus - Kamera-Kommunikationsstatus 1646
3.10 cameratarget – Kameradaten 1648
3.11 capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker 1651
3.12 capdata - CAP-Daten 1653
3.13 caplatrackdata - Track-Daten des CAP-Look-Ahead-Trackers 1657
3.14 capspeeddata - Geschwindigkeitsdaten für CAP 1661
3.15 capstopmode - Definiert die Stopp-Modi für CAP 1663
3.16 captrackdata - CAP-Track-Daten 1664
3.17 capweavedata - Pendeldaten für CAP 1667
3.18 cfgdomain - Konfigurationsdomain 1675
3.19 clock – Zeitmessung 1676
3.20 confdata – Konfigurationsdaten des Roboters 1677
3.21 corrdescr – Deskriptor des Korrekturgenerators 1685
3.22 datapos – Umschließender Block für ein Datenobjekt 1687
3.23 dionum – Digitale Werte (0-1) 1688
3.24 dir – Dateiverzeichnisstruktur 1689
3.25 dnum – Doppelte numerische Werte 1690
3.26 errdomain – Fehlerdomäne 1692
3.27 errnum – Fehlernummer 1694
3.28 errstr – Fehlerreihe 1702
3.29 errtype – Fehlertyp 1703
3.30 event_type – Typ der Ereignisroutine 1704
3.31 exec_level – Abarbeitungsebene 1705
3.32 extjoint - Position von externen Achsen 1706
3.33 flypointdata - Daten für Flying Start/End 1708

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.34 handler_type – Typ des Abarbeitungshandlers 1711
3.35 icondata – Daten für Icon-Anzeige 1712
3.36 identno - Identität für Bewegungsinstruktionen 1714
3.37 intnum - Interrupt-Identität 1716
3.38 iodev – E/A-Gerät 1718
3.39 iounit_state - Status des E/A-Geräts 1719
3.40 jointtarget – Achsenpositionsdaten 1720
3.41 listitem – Datenstruktur des Listenobjekts 1722
3.42 loaddata - Lastdaten 1723
3.43 loadidnum – Lastidentifikationstyp 1729
3.44 loadsession – Ladevorgang für ein Programm 1730
3.45 mecunit – Mechanische Einheit 1731
3.46 motsetdata - Daten für Bewegungseinstellungen 1733
3.47 num - Numerische Werte 1740
3.48 opcalc - Arithmetischer Operator 1742
3.49 opnum - Vergleichsoperator 1743
3.50 orient – Ausrichtung 1744
3.51 paridnum - Parameteridentifikationstyp 1749
3.52 paridvalidnum - Ergebnis von ParIdRobValid 1751
3.53 pathrecid - Bezeichner der Bahnaufzeichnung 1753
3.54 pnpdata – Konfiguration von Aufnahme- und Platzierbahnen 1755
3.55 pos – Positionen (nur X, Y und Z) 1757
3.56 pose – Koordinatentransformationen 1759
3.57 processtimes- Prozesszeiten 1760
3.58 progdisp - Programmverschiebung 1761
3.59 rawbytes – Unverarbeitete Daten 1764
3.60 restartblkdata - Blockdaten für Neustart 1766
3.61 restartdata - Neustartdaten für Trigg-Signale 1768
3.62 rmqheader - RAPID Message Queue-Meldungsheader 1772
3.63 rmqmessage - RAPID Message Queue-Meldung 1774
3.64 rmqslot - Identitätsnummer eines RMQ-Clients 1776
3.65 robjoint – Position der Roboterachsen 1777
3.66 robtarget - Positionsdaten 1778
3.67 sensor - Externer Gerätedeskriptor 1782
3.68 sensorstate - Kommunikationszustand des Geräts 1784
3.69 sensorvardata - Mehrere variable Konfigurationsdaten für die Sensorschnittstelle 1785
3.70 shapedata – Weltzonen-Formdaten 1787
3.71 signalorigin - Gibt die Herkunft des E/A-Signals an 1789
3.72 signalxx - Digitale und analoge Signale 1791
3.73 socketdev - Socketgerät 1793
3.74 socketstatus - Socket-Kommunikationsstatus 1794
3.75 speeddata - Geschwindigkeitsdaten 1795
3.76 stoppointdata - Stopppunktdaten 1799
3.77 string – Zeichenfolgen 1806
3.78 stringdig - Zeichenfolge, die nur Ziffern enthält 1808
3.79 supervtimeouts - Zeitüberschreitungen der Handshake-Überwachung 1809
3.80 switch – Optionale Parameter 1811
3.81 symnum - Symbolische Zahl 1812
3.82 syncident - Identität für Synchronisierungspunkte 1814
3.83 System data – Aktuelle Einstellungen der RAPID-Systemdaten 1815
3.84 taskid – Taskidentifizierung 1818
3.85 tasks – RAPID-Programmtasks 1819
3.86 testsignal – Prüfsignal 1821
3.87 tooldata – Werkzeugdaten 1823
3.88 tpnum - Fensternummer des FlexPendant 1830
3.89 trapdata – Interrupt-Daten für aktuellen TRAP 1831
3.90 triggdata – Positioniereereignisse, trigg 1833
3.91 triggios – Positioniereereignisse, trigg 1835
3.92 triggiosdnum – Positioniereereignisse, trigg 1838

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.93 triggmode - Triggeraktionsmodus 1840
3.94 triggstrgo - Positioning events, trigg 1843
3.95 tsp_status – Status des Taskauswahlfeldes 1846
3.96 tunetype – Servo-Abstimmungstyp 1848
3.97 uishownum - Instanz-ID für UIShow 1849
3.98 weavestartdata - Pendelstartdata 1850
3.99 wobjdata - Werkobjektdaten 1852
3.100 wzstationary - Stationäre Weltzonendaten 1856
3.101 wztemporary - Temporäre Weltzonendaten 1858
3.102 zonedata - Zonendaten 1860

4 Programmierungstyp-Beispiele 1867

4.1 ERROR-Behandlungsroutine mit Bewegungen 1867
4.2 Serviceroutinen mit oder ohne Bewegungen 1870
4.3 System-E/A-Interrupts mit oder ohne Bewegungen 1873
4.4 TRAP-Routinen mit Bewegungen 1876

Index 1879

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Diese Seite wurde absichtlich leer gelassen

Überblick über dieses Handbuch

Über dieses Handbuch

Dies ist ein technisches Referenzhandbuch für den RAPID-Programmierer. In ihm werden die grundlegenden RAPID-Instruktionen, -Funktionen und -Datentypen beschrieben.

Dieses Handbuch beschreibt RobotWare 6.

Verwendung

Das Handbuch sollte während der Programmierung gelesen werden und wenn Sie spezifische Informationen über eine RAPID-Instruktion, eine RAPID-Funktion oder einen RAPID-Datentyp benötigen.

Wer sollte dieses Handbuch lesen?

Dieses Handbuch ist für Personen vorgesehen, die bereits Programmiererfahrung besitzen, z.B. für Roboterprogrammierer.

Voraussetzungen

Der Leser sollte über Programmiererfahrung verfügen und folgende Dokumente gelesen haben:

- *Technisches Referenzhandbuch - RAPID Overview*

Kapiteleinteilung

Dieses Handbuch setzt sich aus den folgenden Kapiteln zusammen:

Kapitel	Inhalt
Instruktionen	Ausführliche Beschreibungen aller grundlegenden RAPID-Instruktionen, einschließlich Verwendungsbeispielen.
Funktionen	Ausführliche Beschreibungen aller grundlegenden RAPID-Funktionen, einschließlich Verwendungsbeispielen.
Datentypen	Ausführliche Beschreibungen aller grundlegenden RAPID-Datentypen, einschließlich Verwendungsbeispielen.
Programmierungstyp-Beispiele	Eine allgemeine Übersicht über das Schreiben von Programmcode, der unterschiedliche Instruktionen/Funktionen/Datentypen enthält. Das Kapitel enthält auch Programmiertipps und Erläuterungen.

Referenzen

Referenz	Dokumentnummer
Technisches Referenzhandbuch - RAPID Overview	*3HAC050947-003*
Technical reference manual - RAPID kernel	*3HAC050946--001*
Anwendungshandbuch - Steuerungssoftware IRC5	*3HAC050798-003*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fortsetzung

Referenz	Dokumentnummer
Bedienungsanleitung - RobotStudio	*3HAC032104-003*

Revisionen

Revision	Beschreibung
-	Veröffentlicht mit RobotWare 6.0. **Hinweis** Für ICR5 mit RobotWare 5, siehe Handbuch3HAC16581-001.
A	Veröffentlicht mit RobotWare 6.01. • Folgende Instruktionen, Funktionen und Datentypen werden hinzugefügt: *AliasIOReset – Zurücksetzen eines E/A-Signals mit Aliasnamen auf Seite 40*, *TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen auf Seite 959* • Informationen zu 7-Achs-Robotern wurden dem Datentyp *confdata – Konfigurationsdaten des Roboters auf Seite 1677*hinzugefügt.
B	Veröffentlicht mit RobotWare 6.02. • Folgende häufige Instruktionen, Funktionen und Datentypen wurden hinzugefügt: *SaveCfgData - Speichert Systemparameter in einer Datei auf Seite 656*, *cfgdomain - Konfigurationsdomain auf Seite 1675*, *TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable auf Seite 919*, *TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen auf Seite 921*, *TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist auf Seite 1563* *AInput - Lesen des Wertes eines analogen Eingangssignals auf Seite 1173*, *DInput - Lesen des Wertes eines digitalen Eingangssignals auf Seite 1284*, *GInput - Liest den Wert des Gruppeneingangssignals. auf Seite 1348* • Alle Anweisungen, Funktionen und Datentypen für die RobotWare-Option *Integrated Vision* hinzugefügt. • Warnung über Bremsweg zu *SoftAct - Aktivieren von Softservo auf Seite 779* hinzugefügt. • Trigonometrische Funktionen für den Datentyp `dnum` hinzugefügt: `ACosDnum, ASinDnum, ATanDnum, ATan2Dnum, CosDnum, TanDnum, SinDnum` • RAPID-Instruktion für die Funktionalität EGM Path Correction hinzugefügt. • Geringfügige Korrekturen.
C	Veröffentlicht mit RobotWare 6.03. • Neue Funktionen zur Instruktion *MotionProcessModeSet - Einstellen des Bewegungsprozess-Modus auf Seite 377* hinzugefügt. • *CAP* Instruktionen, Funktionen, Datentypen hinzugefügt. • *Cyclic bool* Instruktionen und Funktionen hinzugefügt. • Instruktionen und Funktionen im Zusammenhang mit der Funktionalen Sicherheit hinzugefügt. • `signalxx` ist jetzt ein semi-value-Datentyp, der wertorientierte Operationen erlaubt, siehe *signalxx - Digitale und analoge Signale auf Seite 1791*. • Geringfügige Korrekturen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Revision	Beschreibung
D	Veröffentlicht mit RobotWare 6.04. • *SafetyControllerGetOpModePinCode - Pin-Code der Betriebsart abrufen auf Seite 1491* wurde hinzugefügt. • Neue Instruktionen zum Anzeigen der Wartebedingungen hinzugefügt, siehe *UIMsgWrite - Benutzer-Meldungsfenster, Typ nicht-wartend auf Seite 1020*. • Fehlerbehebung `ERR_TASKNAME` hinzugefügt, wo sie gefehlt hat. • Die Robotermodelle *SetLeadThrough - Durchführung aktivieren und deaktivieren auf Seite 717* und *IsLeadThrough - Status der Durchführung prüfen auf Seite 1376* wurden hinzugefügt. • Geringfügige Korrekturen.
E	Veröffentlicht mit RobotWare 6.05. • Neue Instruktion für *Integrated Vision* hinzugefügt, siehe *AliasCamera - Definieren eines Kameragerätes mit Aliasname auf Seite 34*. • Mathematische Instruktionen zur Berechnung von Matrizen hinzugefügt, siehe *MatrixSolve - Lösen eines linearen Gleichungssystems auf Seite 363*. • Neue Instruktionen für Verfahrstrecke und -dauer für die Verfahreinheit hinzugefügt, siehe *ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse auf Seite 613*. • Möglichkeit hinzugefügt, alle `UIxx`- und `TPxx`-Instruktionen mit einer persistenten booleschen Variablen zu unterbrechen. • Neue Instruktionen zur Handhabung mehrerer Variablen von einem Gerät hinzugefügt, siehe *WriteVarArr - Schreiben mehrerer Variablen auf ein Sensorgerät auf Seite 1135*. • Instruktionen `DitherAct` und `DitherDeact` entfernt. • Geringfügige Korrekturen.
F	Veröffentlicht mit RobotWare 6.06. • *FitCircle – Passt einen Kreis an 3D-Punkte an auf Seite 216* wurde hinzugefügt. • *GetJoinData – Abrufen der gelenkspezifischen Daten auf Seite 233* wurde hinzugefügt. • Die Robotermodelle *GetTSPStatus – Rufen Sie den aktuellen Status des Taskauswahlfeldes ab auf Seite 1344* und *tsp_status – Status des Taskauswahlfeldes auf Seite 1846* wurden hinzugefügt. • *IsBrakeCheckActive – Überprüfen Sie, ob die Bremsprüfung ausgeführt wird auf Seite 1366* wurde hinzugefügt. • Die Robotermodelle *TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist auf Seite 1545* und *TaskIsExecuting – Überprüfen Sie, ob die Task ausgeführt wird auf Seite 1547* wurden hinzugefügt. • Geringfügige Korrekturen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fortsetzung

Revision	Beschreibung
G	Veröffentlicht mit RobotWare 6.07. • Funktionen *MaxExtLinearSpeed - Maximale zusätzliche Achsengeschwindigkeit auf Seite 1391*, *MaxExtReorientSpeed - Maximale Zusatzachsen-Rotationsgeschwindigkeit auf Seite 1392* und *MaxRobReorientSpeed - Maximale Umorientierungsgeschwindigkeit des Roboters auf Seite 1393* hinzugefügt. Für *speeddata - Geschwindigkeitsdaten auf Seite 1795* wird der vordefinierte Wert `vmax` geändert, um die von diesen Funktionen zurückgegebenen max. Werte zu verwenden. • Die Funktion *CrossProd - Kreuzprodukte von zwei Positionsvektoren berechnen. auf Seite 1261* wurde hinzugefügt. • Info über Quadranten für 7-Achsen-Roboter zu Abschnitt *confdata – Konfigurationsdaten des Roboters auf Seite 1677* hinzugefügt. • Aktualisierte Informationen zu Einschränkungen für *MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*. • Einschränkungen hinzugefügt für *Comment - Kommentar auf Seite 154*. • Programmausführung bei der Verwendung von `\LockAxis4` in der `SingArea`-Instruktion wurde geklärt. Außerdem wurde die Anmerkung entfernt, die besagt, dass `\LockAxis4` nur für Roboter mit serieller Schnittstelle verwendet werden kann. Siehe *SingArea - Definieren der Interpolation in der Nähe singulärer Punkte auf Seite 736*. • *PathResol - Override der Bahnauflösung auf Seite 534* wurde aktualisiert. • Geringfügige Änderungen der Einschränkungen in *WriteCfgData - Schreibt ein Attribut eines Systemparameters auf Seite 1123*. • Die Instruktionen *CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird. auf Seite 86*, *CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird. auf Seite 89*, *CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird. auf Seite 92* und der Datentyp *capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker auf Seite 1651* wurden hinzugefügt. • Track-Modus 13, 14, und 15 zu dem Datentyp `trackdata`hinzugefügt. • Rotationsreihenfolge in *OrientZYX - Erzeugt eine Orientierung aus Eulerschen Winkeln. auf Seite 1417* und *RelTool - Verschiebung relativ zum Werkzeug auf Seite 1476* verdeutlicht. • Die Instruktionen `EGMStreamStart`, `EGMStreamStop` und `EGMWaitCond` wurden hinzugefügt.
H	Veröffentlicht mit RobotWare 6.08. • Geringfügige Korrekturen. • Aktualisierte Beschränkungen mit Informationen zur Deklaration der Signalvariablen für *AliasIO - Definieren eines E/A-Signals mit Aliasname auf Seite 37* • Die vordefinierte Daten `PRESS_TENDING_MODE` wurden zu *MotionProcessModeSet - Einstellen des Bewegungsprozess-Modus auf Seite 377* hinzugefügt. • Die Instruktion `MovePnP` und der Datentyp `pnpdata` wurden hinzugefügt. • Die Instruktion `SoftElbow` wurde hinzugefügt. • Geringfügige Änderungen an *zonedata - Zonendaten auf Seite 1860*. Referenz zu *Technisches Referenzhandbuch - RAPID Overview*. Dort werden Änderungen für Zonenbahnen ausführlicher beschrieben. • Die Funktion `GetUASUserName` wurde hinzugefügt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Revision	Beschreibung
J	Veröffentlicht mit RobotWare 6.09. • Änderungen an Texten und Beispielen für die Kommunikation vorgenommen, wobei der Schwerpunkt weniger auf seriellen Schnittstellen als auf E/A-Geräten im Allgemeinen liegt. • Beschreibung von `ERR_UNKINO` in Abschnitt *errnum – Fehlernummer auf Seite 1694* aktualisiert. • Aktualisiert Beschreibung über die Signalzugriffsebene in *WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*. • Fehlendes Argument `\StreamStart` wurde zu den Instruktionen `EGMActJoing` und `EGMActPose` hinzugefügt. • Aktualisiert Beschreibung des Arguments `RestartDist` in der Anleitung `CapRefresh`. • Einschränkungsinformationen für mehrere Instruktionen aktualisiert. • Die Instruktionen `WaitSensor`, `WaitWObj` und `Errnum` wurden mit Informationen bezüglich `ERR_CNV_OBJ_LOST` aktualisiert. • Korrigiertes Codebeispiel in *switch – Optionale Parameter auf Seite 1811*. Hinzugefügte Instruktionen: • *CAPSetStopMode - Stellen Sie den Stopp-Modus für Execution Error ein auf Seite 127* • *GetTorqueMargin - Liest die geringste Drehmomentspanne ab auf Seite 1342* • *PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück auf Seite 511* • *PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht auf Seite 513* • *PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht auf Seite 515* • *ResetTorqueMargin - Die geringste Drehmomentspanne zurückstellen auf Seite 619* Hinzugefügte Funktionen: • *PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus auf Seite 1427* Hinzugefügte Datentypen: • *capstopmode - Definiert die Stopp-Modi für CAP auf Seite 1663*
K	Veröffentlicht mit RobotWare 6.10. • Korrektur in Funktion `PathLengthGet`. • Informationen zu IRB 5500 wurde in Abschnitt *ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung auf Seite 156* und *ConfL - Überwachen der Konfiguration während linearer Bewegung auf Seite 158* hinzugefügt. • Die Information zu *Externally Guided Motion* wurde in ein separates Handbuch, *3HAC073319-003*, verschoben.
L	Veröffentlicht mit RobotWare 6.10.02. • Aktualisierte Informationen für `SetAllDataVal` und `SetDataSearch`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fortsetzung

Revision	Beschreibung
M	Veröffentlicht mit RobotWare 6.11. • Aktualisierte Informationen für die Funktion `TextGet`. • Neue Fehlerbehandlung hinzugefügt für Errnum `ERR_CAM_NOT_ON_NETWORK`. • Hinzugefügte Instruktionen: - *SimCollision - Simulation einer Kollision auf Seite 735* - *StrFormat - Formatzeichenfolge. auf Seite 1522* • Hinzugefügte Funktionen: - *CamGetMode - Aktuellen Modus der Kamera abrufen auf Seite 1241* • Hinzugefügte Datentypen: - *camerastatus - Kamera-Kommunikationsstatus auf Seite 1646* • `STIndGun` und `STIndGunReset` aktualisiert mit Servowerkzeuginformationen. • Neue Instruktion `IOEventMessage` hinzugefügt in*IOEventMessage - E/A-Ereignismeldungen von Gerät ein-/ausschalten auf Seite 297*. • Neue Funktion `IsCollFree` hinzugefügt in *IsCollFree - Überprüft die Position auf eine mögliche Kollision auf Seite 1367*. • Funktion `EGMGetState` und Datentypen `egmframetype` und `egmident` aus Handbuch entfernt und verschoben in AM Externally Guided Motion.
N	Veröffentlicht mit RobotWare 6.12. • Hinzugefügte Instruktionen: - *CamStartSetParameter – Start des Setzen-Vorgangs einer Parametereinstellung. auf Seite 74* - *CamWaitSetParameter – Warten, bis ein neuer Set-Vorgang bereit ist. auf Seite 79* • Hinzugefügte Funktionen: - *Rand – Erzeugen Sie eine Zufallszahl auf Seite 1452* • Hinzugefügte Datentypen: - • Informationen aktualisiert zu .xml-Dateien in der Funktion *TextTabGet - Abrufen der Texttabellennummer auf Seite 1561*. • Neue Argumente hinzugefügt (`Arg3` zu `Arg6`) für `StrFormat` und `TextGet`. • Einschränkung hinzugefügt für `StartLoad`, siehe *StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802*. • Informationen aktualisiert zur Verwendung der Instruktion `Stop` in Ereignisroutinen, siehe *Stop - Stoppt die Programmabarbeitung auf Seite 833*.
P	Veröffentlicht mit RobotWare 6.13. • Geringfügige Bildkorrekturen in Abschnitten zu Wait-Instruktionen. • Neues Beispiel in Abschnitt *WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*. • Voraussetzungen hinzugefügt für *GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse auf Seite 1313*.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1 Instruktionen

1.1 AccSet - Verringern der Beschleunigung

Verwendung

`AccSet` wird beim Handhaben von zerbrechlichen Lasten oder zum Verringern von Vibrationen und Bahnfehlern verwendet. Damit ist eine allmählichere Beschleunigung und Verzögerung möglich und die Roboterbewegungen erfolgen fließender.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `AccSet`:

Beispiel 1:

```
AccSet 50, 100;
```

Die Beschleunigung wird auf 50 % ihres normalen Werts beschränkt.

Beispiel 2:

```
AccSet 100, 50;
```

Die Beschleunigungsrampe ist auf 50% des Normalwerts begrenzt, was bedeutet, dass die Zeit, um die Beschleunigung zu erreichen, um den Faktor 2 erhöht wird.

Beispiel 3

```
AccSet 100, 100 \FinePointRamp:=50;
```

Die Verzögerungsrampe wird auf 50 % des normalen Wertes beschränkt, wenn zu einem Feinpunkt hin verzögert wird.

Argumente

```
AccSet Acc Ramp [\FinePointRamp]
```

`Acc`

Datentyp: `num`

Beschleunigung und Verzögerung als Prozentsatz der normalen Werte. 100% entsprechen der maximalen Beschleunigung. Ein Eingangswert <20% ergibt 20% der maximalen Beschleunigung.

`Ramp`

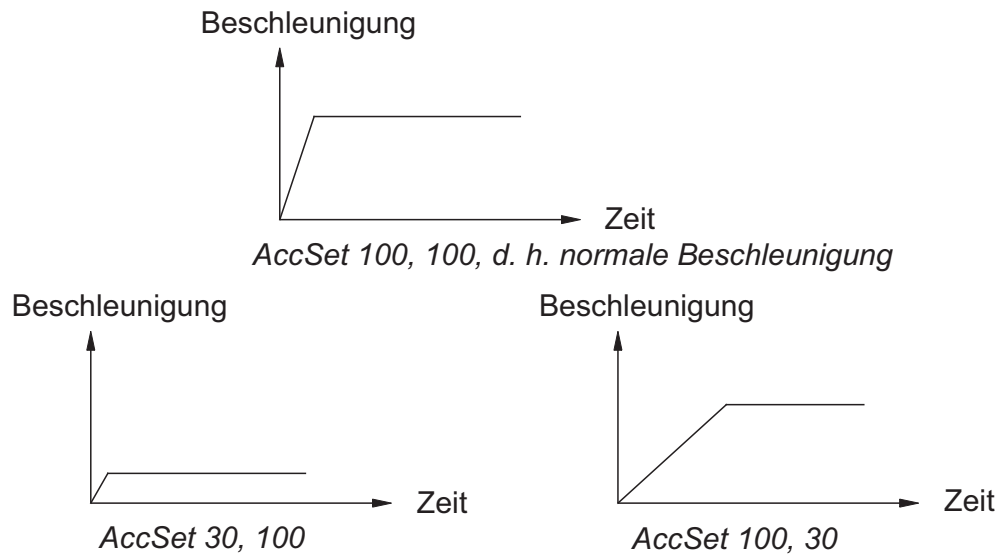
Datentyp: `num`

Anstiegswert von Beschleunigung und Verzögerung als Prozentsatz der normalen Werte. Erschütterungen können durch Verringern dieses Werts begrenzt werden. 100% entspricht der maximalen Rate. Ein Eingangswert <10% ergibt 10% der maximalen Rate.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Zahlen zeigen, dass eine Verringerung der Beschleunigung fließendere Bewegungen bewirkt.



xx0500002146

`[\FinePointRamp]`

Datentyp: `num`

Wert, zu welchem die Verzögerung als Prozentsatz der normalen Werte abnimmt. Der Parameter wirkt sich nur auf die Rampe aus, wenn sich der Roboter zu einem Feinpunkt hin verzögert. Bei einem Feinpunkt ist der Wert der Verzögerungsrampe eine Kombination aus diesem Parameter und dem `Ramp` Wert, `Ramp` * `FinePointRamp`. Der Parameter muss größer als 0 sein und in dem Intervall zwischen 0 und 100 % liegen.

Wenn dieses optionale Argument nicht verwendet wird, wird der `FinePointRamp` Wert auf den Standardwert, 100 %, gesetzt.

Programmabarbeitung

Die Beschleunigung gilt die als nächstes ausgeführte Bewegungsinstruktion, und zwar sowohl für den Roboter als die externen Achsen, bis ein neue `AccSet`-Instruktion abgearbeitet wird.

Die Standardwerte (`AccSet 100, 100`) werden automatisch gesetzt

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Syntax

```
AccSet
  [ Acc ':=' ] < expression (IN) of num > ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[ Ramp ':=' ] < expression (IN) of num >
[ '\'FinePointRamp ':=' < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Beschränken der TCP-Beschleunigung entlang der Bahn	*PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn auf Seite 507*
Definition der Höchstgeschwindigkeit	*VelSet - Ändern der programmierten Geschwindigkeit auf Seite 1038*
Steuern der Beschleunigung im Welt-Koordinatensystem	*WorldAccLim - Steuern der Beschleunigung im Welt-Koordinatensystem auf Seite 1112*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.2 ActEventBuffer – Aktivierung des Ereignispuffers

Beschreibung

`ActEventBuffer` wird verwendet, um die Verwendung des Ereignispuffers in der aktuellen Bewegungsprogrammtask zu aktivieren.

Die Instruktionen `ActEventBuffer` und `DeactEventBuffer` sollten verwendet werden, wenn eine Anwendung mit Feinpunkten und eine fortlaufende Anwendung, bei der Signale aufgrund der langsamen Prozessausrüstung vorher gesetzt werden müssen, kombiniert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ActEventBuffer`:

Beispiel 1:

```
...
DeactEventBuffer;
! Use an application that uses finepoints, such as SpotWelding
...
! Activate the event buffer again
ActEventBuffer;
! Now it is possible to use an application that needs
! to set signals in advance, such as Dispense
...
```

Der `DeactEventBuffer` deaktiviert den konfigurierten Ereignispuffer. Bei einer Anwendung mit Feinpunkten findet der Roboterstart am Feinpunkt schneller statt. Wird der Ereignispuffer mit `ActEventBuffer` aktiviert, können bei einer Anwendung mit langsamer Prozessausrüstung Signale im Voraus festgelegt werden.

Programmabarbeitung

Die Verwendung eines Ereignispuffers gilt jeweils für die nächste abgearbeitete Roboterinstruktion und ist so lange wirksam, bis eine `DeactEventBuffer`-Instruktion ausgeführt wird.

Die Instruktion wartet mit der Aktivierung des Ereignispuffers, bis der Roboter und die externen Achsen den Stopppunkt (`ToPoint` der aktuellen Bewegungsinstruktion) erreicht haben. Deshalb wird empfohlen, die Bewegungsinstruktion vor `ActEventBuffer` mit einem Feinpunkt zu programmieren.

Der Standardwert (`ActEventBuffer`) wird automatisch gesetzt

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

`ActEventBuffer` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

Syntax

```
ActEventBuffer ';'
```

Weitere Informationen

Information über	Siehe
Deaktivierung des Ereignispuffers	*DeactEventBuffer – Deaktivierung des Ereignispuffers auf Seite 185*
Konfiguration von Event preset time	*Technisches Referenzhandbuch - Systemparameter*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.3 ActUnit - Aktivieren einer mechanischen Einheit

Verwendung

`ActUnit` wird verwendet, um eine mechanische Einheit zu aktivieren.

Mit dieser Instruktion kann auch bestimmt werden, welche Einheit aktiv ist, wenn gemeinsame Antriebseinheiten verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ActUnit`:

Beispiel 1:

```
ActUnit orbit_a;
```

Aktivieren der mechanischen Einheit `orbit_a`.

Argumente

```
ActUnit MechUnit
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, die aktiviert werden soll.

Programmabarbeitung

Wenn die Roboter und die tatsächliche Bahn der externen Achsen bereit sind, wird die Bahn auf der aktuellen Bahnebene gelöscht und die angegebene mechanische Einheit wird aktiviert. Dies bedeutet, dass sie vom Roboter gesteuert und überwacht wird.

Wenn mehrere mechanische Einheiten eine gemeinsame Antriebseinheit nutzen, wird diese Einheit beim Aktivieren einer dieser mechanischen Einheiten auch mit der gemeinsamen Antriebseinheit verbunden.

Einschränkungen

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden, anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`ActUnit` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

`ActUnit` - `DeactUnit` kann auf der `StorePath`-Ebene verwendet werden, doch müssen bei der Ausführung von `RestoPath` dieselben mechanischen Einheiten wie bei der Ausführung von `StorePath` aktiviert sein. Bei einer solchen Operation sind die Bahnaufzeichnung und die Bahn auf der Basisebene nicht betroffen, doch die Bahn auf der `StorePath`-Ebene wird gelöscht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
ActUnit
  [ MechUnit ':=' ] < variable (VAR) of mecunit> ';'
```

Weitere Informationen

Information über	Siehe
Deaktivieren mechanischer Einheiten	*DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Mechanische Einheiten	*mecunit – Mechanische Einheit auf Seite 1731*
Weitere Beispiele	*DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Prüft, ob eine mechanische Einheit aktiviert ist.	*IsMechUnitActive - Prüfen, ob mechanische Einheit aktiv ist auf Seite 1378*
Bahnaufzeichnung	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.4 Add – Addiert einen numerischen Wert

Verwendung

`Add` wird verwendet, um einen Wert zu einer numerischen Variablen oder Persistenten zu addieren oder von ihr zu subtrahieren.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Add`:

Beispiel 1:

```
Add reg1, 3;
```

`3` wird zu `reg1` addiert, das ergibt `reg1:=reg1+3`.

Beispiel 2:

```
Add reg1, -reg2;
```

Der Wert von `reg2` wird von `reg1` subtrahiert, das ergibt `reg1:=reg1-reg2`.

Beispiel 3

```
VAR dnum mydnum:=5;
Add mydnum, 500000000;
```

`500000000` wird zu `mydnum` addiert, das ergibt mynum:=mynum+500000000.

Beispiel 4:

```
VAR dnum mydnum:=5000;
VAR num mynum:=6000;
Add mynum, DnumToNum(mydnum \Integer);
```

`5000` wird zu `mynum` addiert, das ergibt `mynum:=mynum+5000`. Sie müssen `DnumToNum` verwenden, um einen numerischen `num`-Wert zu erhalten, den Sie zusammen mit der `num`-Variable `mynum` einsetzen können.

Argumente

```
Add Name | Dname AddValue | AddDvalue
```

`Name`

Datentyp: `num`

Der Name der zu ändernden Variablen oder Persistenten.

`Dname`

Datentyp: `dnum`

Der Name der zu ändernden Variablen oder Persistenten.

`AddValue`

Datentyp: `num`

Der zu addierende Wert.

`AddDvalue`

Datentyp: `dnum`

Der zu addierende Wert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Wenn der hinzugefügte Wert vom Typ `dnum` ist und die zu ändernde Variable/Persistente ein `num` ist, wird ein Laufzeitfehler generiert. Argumente können nicht kombiniert werden (zur Lösung dieses Problems siehe weiter oben Beispiel 4).

Syntax

```
Add
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname ':=' ] < var or pers (INOUT) of dnum > ','
  [ AddValue ':=' ] < expression (IN) of num >
  | [ AddDvalue ':=' ] < expression (IN) of dnum > ';'
```

Weitere Informationen

Information über	Siehe
Inkrementieren der Variablen um 1	*Incr - Inkrementieren um 1 auf Seite 260*
Dekrementieren der Variable um 1	*Decr - Dekrementieren um 1 auf Seite 189*
Ändern von Daten über einen beliebigen Ausdruck, z. B. Multiplikation	*":=" - Zuweisen eines Werts auf Seite 42*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.5 AliasCamera - Definieren eines Kamerageräts mit Aliasname

Verwendung

`AliasCamera` wird verwendet, um eine Kamera mit einem Aliasnamen zu definieren oder um Signale in integrierten Taskmodulen zu verwenden. Kameras mit Aliasnamen können für vordefinierte generische Programme verwendet werden. Die Instruktion `AliasCamera` muss vor jeder Verwendung der eigentlichen Kamera ausgeführt werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `AliasCamera`.

Beispiel 1:

```
VAR cameradev mycamera;
...
PROC prog_start()
  AliasCamera "CAMERA1", mycamera;
  ...
  CamReqImage mycamera;
```

Die Routine `prog_start` wird zu Beginn des RAPID-Programms ausgeführt. Instruktion `AliasCamera` sucht nach der vordefinierten Variable des RAPID-Kamerageräts mit dem Namen `CAMERA1`, und deren Inhalt wird nach `mycamera` kopiert. Ab sofort ist es möglich, auf die Kamera mit dem `mycamera` Kameragerät zuzugreifen.

Beispiel 2:

```
VAR cameradev mycamera;
PROC proc1()
  IF GetTaskName() = "T_ROB_L" THEN
    AliasCamera CAMERA_L, mycamera;
  ELSE
    AliasCamera CAMERA_R, mycamera;
  ENDIF
  ...
  CamReqImage mycamera;
```

Die Routine `proc1` ist mit dem `START`-Ereignis in den Systemparametern verbunden. Das Programm, welche das Kameragerät `mycamera` definiert, ist beim Programmstart mit der konfigurierten Kamera `CAMERA_L` oder `CAMERA_R` verbunden.

Argumente

```
AliasCamera CameraName | FromCamera ToCamera
```

`CameraName`

Datentyp: `string`

Der Kamerabezeichner gemäß der Konfiguration des Systemparameters Communication. Die Instruktion `AliasCamera` sucht nach der vordefinierten Variable des RAPID-Kamerageräts (installierte Daten) mit dem Namen, der in `CameraName` verwendet wird, und kopiert deren Inhalt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`FromCamera`

Datentyp: `cameradev`

Der Kamerabezeichner in der Konfiguration des Systemparameters Communication, aus der das Kameragerät kopiert wird. Die Kamera muss in den Systemparametern definiert werden.

`ToCamera`

Datentyp: `cameradev`

Der Kamerabezeichner entsprechend dem Programm, in welches das Kameragerät kopiert wird. Die `cameradev` muss im RAPID-Programm deklariert werden.

Programmabarbeitung

Der Inhalt des Kamerageräts wird von der in Argument `CameraName` oder `FromCamera` angegebenen Kamera zu dem Kameragerät in Argument `ToCamera` kopiert.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_ALIASCAM_DEF`	Die Kamera in Argument `CameraName` oder der/die/das `cameradev` in Argument `FromCamera` ist nicht in der Konfiguration des Systemparameters Communication definiert. Oder der/die/das `ToCamera` ist nicht im RAPID-Programm deklariert oder bereits in der Konfiguration des Systemparameters Communication definiert.

Einschränkung

Wenn das Programm gestartet wird, kann das Aliassignal nicht verwendet werden, bis die Instruktion `AliasCamera` ausgeführt wird.

Die Instruktion `AliasCamera` **muss** folgendermaßen platziert werden:

- entweder in der beim Programmstart ausgeführten Ereignisroutine (Ereignis `START`).
- oder in dem Programmteil, der nach jedem Programmstart ausgeführt wird (bevor die Kamera verwendet wird).

Um Fehler zu vermeiden, ist es nicht empfehlenswert, für eine `AliasCamera` Kamera eine dynamische Wiederherstellung der Verbindung zu verschiedenen physikalischen Kameras zu verwenden.

Syntax

```
AliasCamera
  [ CameraName ':=' ] < expression (IN) of string >
  | FromCamera ':=' < variable (VAR) of cameradev > ','
  [ ToCamera ':=' < variable (VAR) of cameradev > ] ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.5 AliasCamera - Definieren eines Kamerageräts mit Aliasname

Integrated Vision

Fortsetzung

Weitere Informationen

Information über	Siehe
Definition von Kameragerätеn	*cameradev – Kameragerät auf Seite 1645*
Konfiguration der Kameras	*Technisches Referenzhandbuch - Systemparameter*
Definieren von Ereignisroutinen	*Technisches Referenzhandbuch - Systemparameter*
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.6 AliasIO - Definieren eines E/A-Signals mit Aliasname

Verwendung

`AliasIO` wird verwendet, um ein Signal beliebigen Typs mit einem Aliasnamen zu definieren oder um Signale in integrierten Taskmodulen zu verwenden.

Signale mit Aliasnamen können für vordefinierte generische Programme verwendet werden, ohne Änderungen am Programm vornehmen zu müssen, bevor es in einer anderen Roboterinstallation ausgeführt werden kann.

Die Instruktion `AliasIO` muss vor jeder Verwendung des tatsächlichen Signals ausgeführt werden Siehe *Grundlegende Beispiele auf Seite 37* für geladene Module und *Weitere Beispiele auf Seite 38* für installierte Module.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `AliasIO`:

Siehe auch *Weitere Beispiele auf Seite 38*.

Beispiel 1:

```
VAR signaldo alias_do;
PROC prog_start()
  AliasIO config_do, alias_do;
ENDPROC
```

Die Routine `prog_start` ist mit dem `START`-Ereignis in den Systemparametern verknüpft. Das durch das Programm definierte digitale Ausgangssignal `alias_do` ist beim Programmstart mit dem konfigurierten digitalen Ausgangssignal `config_do` verknüpft.

Argumente

```
AliasIO FromSignal ToSignal
```

`FromSignal`

Datentyp: `signalxx` oder `string`

Geladene Module:

Der Signalbezeichner, der entsprechend der Konfiguration (Datentyp `signalxx`) benannt wurde, aus der der Signaldeskriptor kopiert wird. Das Signal muss in der E/A-Konfiguration definiert sein.

Installierte Module oder geladene Module:

Eine Referenz (`CONST`, `VAR` oder ein Parameter davon) mit dem Namen des Signals (Datentyp `string`), aus der der Signaldeskriptor nach einer Suche im System kopiert wird. Das Signal muss in der E/A-Konfiguration definiert sein.

`ToSignal`

Datentyp: `signalxx`

Der Signalbezeichner gemäß dem Programm (Datentyp `signalxx`), in das der Signaldeskriptor kopiert wird. Das Signal muss im RAPID-Programm deklariert sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Für die Argumente `FromSignal` und `ToSignal` muss derselbe Datentyp verwendet (oder gesucht) werden. Dies muss der Typ `signalxx` (`signalai`, `signalao`, `signaldi`, `signaldo`, `signalgi` oder `signalgo`) sein.

Programmabarbeitung

Der Wert des Signaldeskriptors wird aus dem im Argument `FromSignal` angegebenen Signal in das im Argument `ToSignal` angegebene Signal kopiert.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ALIASIO_DEF`	`FromSignal` ist nicht in der E/A-Konfiguration definiert oder `ToSignal` ist nicht im RAPID-Programm deklariert oder `ToSignal` ist nicht in der E/A-Konfiguration definiert.
`ERR_ALIASIO_TYPE`	Die Datentypen für die Argumente `FromSignal` und `ToSignal` sind nicht identisch.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.

Weitere Beispiele

Weitere Beispiele für die Instruktion `AliasIO` sind unten aufgeführt.

Beispiel 1:

```
VAR signaldi alias_di;
PROC prog_start()
  CONST string config_string := "config_di";
  AliasIO config_string, alias_di;
ENDPROC
```

Die Routine `prog_start` ist mit dem `START`-Ereignis in den Systemparametern verknüpft. Das durch das Programm definierte digitale Eingangssignal `alias_di` ist mit dem konfigurierten digitalen Eingangssignal `config_di` bei Programmstart verknüpft (über die Konstante `config_string`).

Einschränkungen

Wenn das Programm gestartet wird, kann das Aliassignal nicht verwendet werden, bis die Instruktion `AliasIO` ausgeführt wird.

Die Signalvariable sollte im Modul global deklariert werden. Sie darf nicht Teil einer RECORD-Komponente sein oder lokal in einer Prozedur deklariert werden (andernfalls funktioniert die Aktualisierung des Signals nach einem Stromausfall nicht wie gewünscht).

Instruktion `AliasIO` **muss** angebracht werden

- entweder in der beim Programmstart ausgeführten Ereignisroutine (Ereignis `START`),

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- oder in dem Programmteil, der nach jedem Programmstart ausgeführt wird (bevor das Signal verwendet wird).

Um Fehler zu vermeiden, ist es nicht empfehlenswert, für ein `AliasIO`-Signal eine dynamische Wiederherstellung der Verbindung zu verschiedenen physikalischen Signalen zu verwenden.

Syntax

```
AliasIO
  [ FromSignal ':=' ] < reference (REF) of anytype > ','
  [ ToSignal ':=' ] < variable (VAR) of anytype > ';'
```

Weitere Informationen

Information über	Siehe
Zurücksetzen eines E/A-Signals mit Aliasnamen	*AliasIOReset – Zurücksetzen eines E/A-Signals mit Aliasnamen auf Seite 40*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Abruf von Informationen zur Herkunft eines E/A-Signals.	*GetSignalOrigin - Abruf von Informationen zur Herkunft eines E/A-Signals. auf Seite 1333*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*
Definieren von Ereignisroutinen	*Technisches Referenzhandbuch - Systemparameter*
Geladene/Installierte Taskmodule	*Technisches Referenzhandbuch - Systemparameter*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.7 AliasIOReset – Zurücksetzen eines E/A-Signals mit Aliasnamen

Verwendung

`AliasIOReset` wird verwendet, um ein Signal zurückzusetzen, das in einem vorhergehenden Aufruf von `AliasIO` verwendet wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `AliasIOReset`:

Beispiel 1:

```
VAR signaldo alias_do;
PROC myproc()
  AliasIO config_do, alias_do;
  SetDO alias_do, 1;
  ..
  AliasIOReset alias_do;
ENDPROC
```

Das Programm, welches das digitale Ausgangssignal `alias_do` definiert, ist mit dem konfigurierten digitalen Ausgangssignal `config_do` zu Beginn der Prozedur `myproc` verbunden. Das Signal `config_do` wird in der E/A-Konfiguration definiert. Später, wenn `alias_do` nicht mehr verwendet werden darf, wird die Alias-Verkopplung entfernt.

Argumente

```
AliasIOReset Signal
```

`Signal`

Datentyp: `signalxx`

Der Signalbezeichner gemäß dem Programm (Datentyp `signalxx`), das zurückgesetzt werden muss. Das Signal muss im RAPID-Programm deklariert sein.

Programmabarbeitung

Die Alias-Verkopplung wurde vollständig entfernt. Das Signal kann nicht verwendet werden, bevor eine neue Alias-Verkopplung mit `AliasIO` erfolgt.

Einschränkung

Signale, die in der E/A-Konfiguration definiert wurden, können nicht zurückgesetzt werden. Es können nur Signale verwendet werden, die in einer `AliasIO`-Instruktion verwendet wurden und im RAPID-Programm deklariert sind.

Syntax

```
AliasIOReset
  [ Signal ':=' ] < variable (VAR) of anytype > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Definieren eines E/A-Signals mit Aliasname	*AliasIO - Definieren eines E/A-Signals mit Aliasname auf Seite 37*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Abruf von Informationen zur Herkunft eines E/A-Signals	*GetSignalOrigin - Abruf von Informationen zur Herkunft eines E/A-Signals. auf Seite 1333*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*
Definieren von Ereignisroutinen	*Technisches Referenzhandbuch - Systemparameter*
Geladene/Installierte Taskmodule	*Technisches Referenzhandbuch - Systemparameter*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.8 ":=" - Zuweisen eines Werts

Verwendung

Mit der Instruktion „:=“ wird Daten ein neuer Wert zugewiesen. Es kann ein beliebiger Wert von einer Konstanten bis zu einem arithmetischen Ausdruck sein, z. B. `reg1+5*reg3`.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion „:=“:

Siehe auch *Weitere Beispiele auf Seite 42*.

Beispiel 1:

```
reg1 := 5;
```

`reg1` wird der Wert `5` zugewiesen.

Beispiel 2:

```
reg1 := reg2 - reg3;
```

`reg1` wird der Wert zugewiesen, den die Berechnung `reg2-reg3` zurückgibt.

Beispiel 3

```
counter := counter + 1;
```

`counter` wird um 1 inkrementiert.

Argumente

```
Data := Value
```

`Data`

Datentyp: `All`

Die Daten, denen ein neuer Wert zugewiesen werden soll.

`Value`

Datentyp: `Same as Data`

Der gewünschte Wert.

Weitere Beispiele

Weitere Beispiele für die Instruktion ":=" sind unten aufgeführt.

Beispiel 1:

```
tool1.tframe.trans.x := tool1.tframe.trans.x + 20;
```

Der TCP für `tool1` wird um `20` mm in X-Richtung verschoben.

Beispiel 2:

```
pallet{5,8} := Abs(value);
```

Einem Element in der Matrix `pallet` wird ein Wert zugewiesen, der gleich dem Absolutwert der Variablen `value` ist.

Einschränkungen

Die Daten (deren Wert geändert werden soll) dürfen nicht Folgendes sein:

- eine Konstante,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Datentyp ohne einen Wert (Non-Value).

Daten und Wert müssen ähnliche Datentypen (identisch oder Alias) aufweisen.

Syntax

```
<assignment target> ':=' <expression> ';'
```

Weitere Informationen

Information über	Siehe
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*
Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*
Zuweisen eines Anfangswerts für Daten	*Bedienungsanleitung - IRC5 mit FlexPendant*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.9 BitClear – Löschen eines angegebenen Bits in Byte- oder dnum-Daten

Verwendung

`BitClear` wird verwendet, um ein angegebenes Bit in definierten `byte`- oder `dnum`-Daten zu löschen (auf 0 zu setzen).

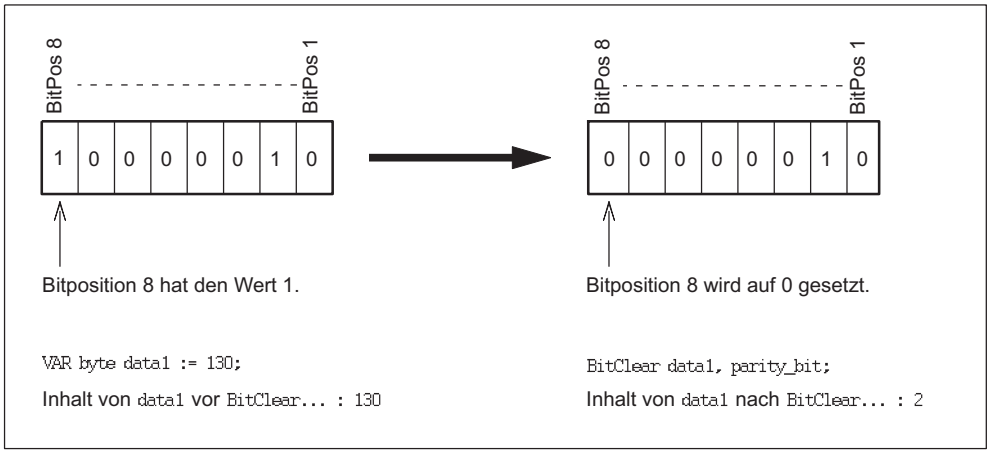
Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `BitClear`:

Beispiel 1:

```
CONST num parity_bit := 8;
VAR byte data1 := 130;
BitClear data1, parity_bit;
```

Die Bitnummer `8` (parity_bit) in der Variablen `data1` wird auf 0 gesetzt, z. B. wird der Inhalt der Variablen `data1` von 130 auf 2 geändert (ganzzahlige Darstellung). Die Bitbearbeitung von Datentyp `byte` bei Verwendung von `BitClear` wird in der folgenden Abbildung gezeigt.



xx0500002147

Beispiel 2:

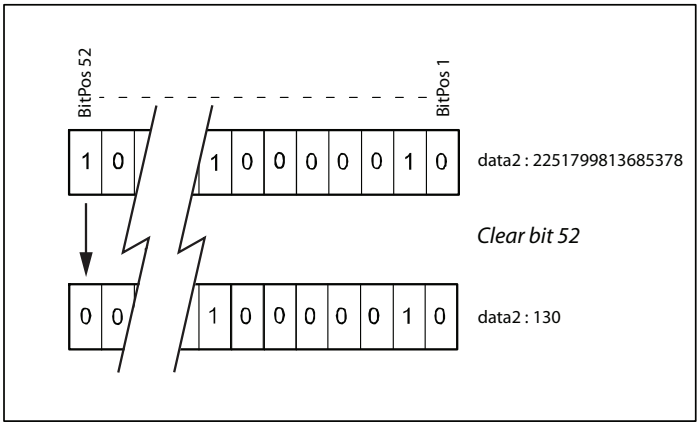
```
CONST num parity_bit := 52;
VAR dnum data2 := 2251799813685378;
BitClear data2, parity_bit;
```

Bit Nummer `52` (parity_bit) in der Variablen `data2` wird auf 0 gesetzt, z. B. wird der Inhalt der Variablen `data2` von 2251799813685378 in 130 geändert (ganzzahlige

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Darstellung). Die Bitmanipulation des Datentyps `dnum` bei Verwendung von `BitClear` wird in der nachfolgenden Abbildung gezeigt.



xx1200000014

Argumente

```
BitClear BitData | DnumData BitPos
```

`BitData`

Datentyp: `byte`

Die zu ändernden Bitdaten in ganzzahliger Darstellung.

`DnumData`

Datentyp: `dnum`

Die zu ändernden dnum-Bitdaten in ganzzahliger Darstellung.

`BitPos`

Bit Position

Datentyp: `num`

Die auf 0 zu setzende Bitposition (1-8) in den `BitData` oder (1-52) in den `DnumData`.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255 dezimal.

Für die Bitposition beim Datentyp `byte` ist ein Wert von 1-8 gültig.

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495 dezimal.

Für die Bitposition beim Datentyp `dnum` ist ein Wert von 1-52 gültig.

Syntax

```
BitClear
  [ BitData ':=' ] < var or pers (INOUT) of byte >
  | [ DnumData ':=' ] < var or pers (INOUT) of dnum > ','
  [ BitPos ':=' ] < expression (IN) of num > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.9 BitClear – Löschen eines angegebenen Bits in Byte- oder dnum-Daten
RobotWare - OS
Fortsetzung

Weitere Informationen

Information über	Siehe
Setzen eines angegebenen Bits in Byte- oder dnum-Daten	*BitSet – Setzen eines angegebenen Bits in Byte- oder dnum-Daten auf Seite 47*
Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist	*BitCheck - Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist auf Seite 1193*
Prüfen, ob ein angegebenes Bit in dnum-Daten gesetzt ist	*BitCheckDnum – Prüfen, ob ein angegebenes Bit in den dnum-Daten gesetzt ist auf Seite 1195*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.10 BitSet – Setzen eines angegebenen Bits in Byte- oder dnum-Daten

Verwendung

Mit `BitSet` wird ein bestimmtes Bit in definierten `byte`- oder `dnum`-Daten auf 1 gesetzt.

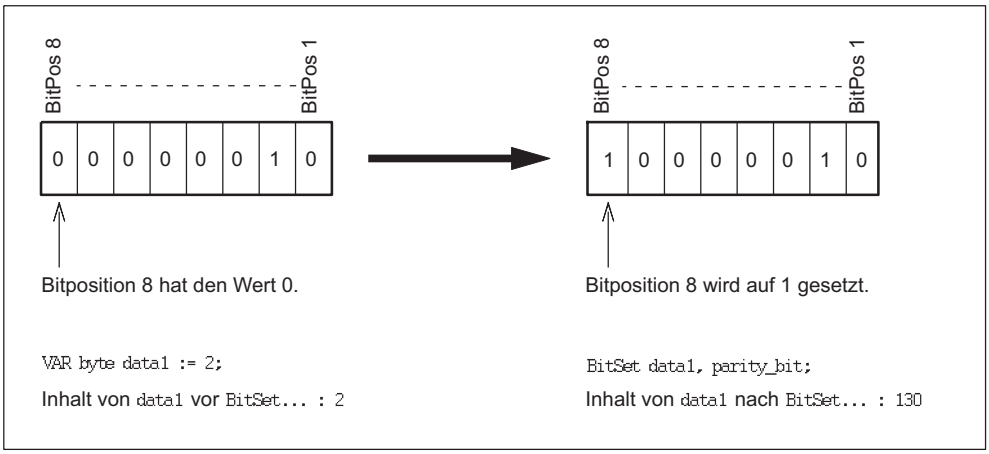
Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `BitSet`:

Beispiel 1:

```
CONST num parity_bit := 8;
VAR byte data1 := 2;
BitSet data1, parity_bit;
```

Die Bitnummer 8 (`parity_bit`) in der Variablen `data1` wird auf `1` gesetzt, z. B. wird der Inhalt der Variablen `data1` von 2 auf 130 geändert (ganzzahlige Darstellung). Die Bitbearbeitung von Datentyp `byte` bei Verwendung von `BitSet` wird in der Abbildung unten gezeigt.



xx0500002148

Beispiel 2:

```
CONST num parity_bit := 52;
VAR dnum data2 := 130;
BitSet data2, parity_bit;
```

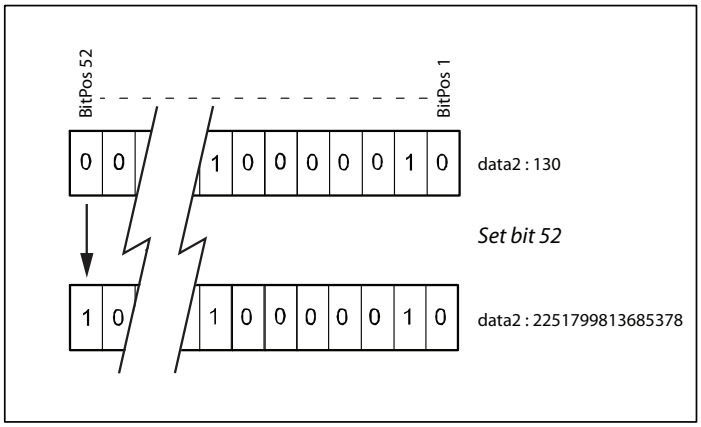
Bit Nummer 52 (`parity_bit`) in der Variablen `data2` wird auf `1` gesetzt, z. B. wird der Inhalt der Variablen `data2` von 130 in 2251799813685378 geändert (ganzzahlige

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.10 BitSet – Setzen eines angegebenen Bits in Byte- oder dnum-Daten
RobotWare - OS
Fortsetzung

Darstellung). Die Bitmanipulation des Datentyps `dnum` bei Verwendung von `BitSet` wird in der nachfolgenden Abbildung gezeigt.



xx1200000015

Argumente

```
BitSet BitData | DnumData BitPos
```

`BitData`

Datentyp: `byte`

Die zu ändernden Bitdaten in ganzzahliger Darstellung.

`DnumData`

Datentyp: `dnum`

Die zu ändernden Bitdaten in ganzzahliger Darstellung.

`BitPos`

Bit Position

Datentyp: `num`

Die auf 1 zu setzende Bitposition (1-8) in den `BitData` oder (1-52) in den `DnumData`.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255 (ganzzahlig).

Für die Bitposition beim Datentyp `byte` ist ein Wert von 1-8 gültig.

Der Bereich für den Datentyp `dnum` ist 0 bis 4503599627370495 (ganzzahlig).

Für die Bitposition beim Datentyp `dnum` ist ein Wert von 1-52 gültig.

Syntax

```
BitSet
  [ BitData':=' ] < var or pers (INOUT) of byte >
  | [ DnumData':=' ] < var or pers (INOUT) of dnum > ','
  [ BitPos':=' ] < expression (IN) of num > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Löschen eines angegebenen Bits in Byte- oder dnum-Daten	*BitClear – Löschen eines angegebenen Bits in Byte- oder dnum-Daten auf Seite 44*
Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist	*BitCheck - Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist auf Seite 1193*
Prüfen, ob ein angegebenes Bit in dnum-Daten gesetzt ist	*BitCheckDnum – Prüfen, ob ein angegebenes Bit in den dnum-Daten gesetzt ist auf Seite 1195*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.11 BookErrNo - Vergeben einer RAPID-Systemfehlernummer

Verwendung

`BookErrNo` wird verwendet, um eine neue RAPID-Systemfehlernummer zu vergeben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `BookErrNo`:

Beispiel 1:

```
! Introduce a new error number in a glue system
! Note: The new error variable must be declared with the initial
     value -1
VAR errnum ERR_GLUEFLOW := -1;

! Book the new RAPID system error number
BookErrNo ERR_GLUEFLOW;
```

Die Variable `ERR_GLUEFLOW` wird zur Verwendung im RAPID-Code einer freien Systemfehlernummer zugewiesen.

```
! Use the new error number
IF di1 = 0 THEN
  RAISE ERR_GLUEFLOW;
ELSE
...
ENDIF

! Error handling
ERROR
  IF ERRNO = ERR_GLUEFLOW THEN
    ...
  ELSE
    ...
  ENDIF
```

Bei einer digitalen Eingabe `di1` mit dem Wert `0` wird die neu vergebene Fehlernummer ausgelöst und die Systemfehlervariable `ERRNO` wird auf den Wert der neu vergebenen Fehlernummer gesetzt. Die Fehlerbehandlung dieser benutzergenerierten Fehler kann anschließend wie üblich erfolgen.

Argumente

```
BookErrNo ErrorName
```

`ErrorName`

Datentyp: `errnum`

Der neue Name der RAPID-Systemfehlervariablen.

Einschränkungen

Die neue Fehlervariable darf nicht als Routinenvariable deklariert werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die neue Fehlervariable muss mit einem Anfangswert von -1 deklariert werden. Dadurch wird angegeben, dass es sich bei diesem Fehler um einen RAPID-Systemfehler handelt.

Syntax

```
BookErrNo
  [ ErrorName ':='] < variable (VAR) of errnum > ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview*
Fehlernummer	*errnum – Fehlernummer auf Seite 1694*
Aufrufen einer Fehlerbehandlung	*RAISE - Ruft die Fehlerbearbeitung auf auf Seite 575*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.12 Break - Abbruch der Programmabarbeitung

Verwendung

`Break` wird verwendet, um die Programmabarbeitung abzubrechen und sofort nach Fehlern im RAPID-Programmcode zu suchen. Die Roboterbewegung wird sofort beendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Break`:

Beispiel 1:

```
...
Break;
...
```

Die Programmabarbeitung stoppt und Sie können Variablen, Werte usw. zur Fehlerbeseitigung analysieren.

Programmabarbeitung

Die Instruktion stoppt die Programmabarbeitung sofort, ohne darauf zu warten, dass Roboter und externe Achsen ihre programmierten Zielpunkte für die aktuelle Bewegung erreichen. Die Programmabarbeitung kann dann ab der nächsten Instruktion neu gestartet werden.

Falls in einer Ereignisroutine die Instruktion `Break` enthalten ist, wird die Abarbeitung der Routine unterbrochen und es wird keine STOP-Ereignisroutine ausgeführt. Die Ereignisroutine wird beim nächsten Auftreten desselben Ereignisses von Anfang an abgearbeitet.

Syntax

```
Break';'
```

Weitere Informationen

Information über	Siehe
Stoppen für Programmaktionen	*Stop - Stoppt die Programmabarbeitung auf Seite 833*
Stoppen nach einem schwerwiegenden Fehler	*EXIT - Beenden der Programmabarbeitung auf Seite 213*
Beenden der Programmabarbeitung	*EXIT - Beenden der Programmabarbeitung auf Seite 213*
Nur Roboterbewegungen stoppen	*StopMove - Stoppen der Roboterbewegung auf Seite 839*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.13 CallByVar - Aufruf einer Prozedur durch eine Variable

Verwendung

`CallByVar` (*Call By Variable*) dient dem Aufruf von Prozeduren mit bestimmten Namen, z. B. `proc_name1, proc_name2, proc_name3 ... proc_namex` mit einer Variablen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CallByVar`:

Siehe auch *Weitere Beispiele auf Seite 53*.

Beispiel 1:

```
reg1 := 2;
CallByVar "proc", reg1;
```

Die Prozedur `proc2` wird aufgerufen.

Argumente

```
CallByVar Name Number
```

`Name`

Datentyp: `string`

Der erste Teil des Prozedurnamens, z. B. `proc_name`.

`Number`

Datentyp: `num`

Der numerische Wert für die Nummer der Prozedur. Dieser Wert wird in eine Zeichenfolge konvertiert und liefert den zweiten Teil des Prozedurnamens, z. B. `1`. Der Wert muss eine positive Ganzzahl sein.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_ARGVALERR	Das Argument `Number` ist < 0 oder keine ganze Zahl.
ERR_REFUNKPRC	Der Verweis gilt für eine unbekannte Prozedur.
ERR_CALLPROC	Fehler bei Prozeduraufruf (nicht bei der Prozedur selbst).

Weitere Beispiele

Weitere Beispiele für die statische und dynamische Auswahl eines Prozeduraufrufs.

Beispiel 1 - Statische Auswahl eines Prozeduraufrufs

```
TEST reg1
  CASE 1:
    lf_door door_loc;
  CASE 2:
    rf_door door_loc;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.13 CallByVar - Aufruf einer Prozedur durch eine Variable

RobotWare - OS

Fortsetzung

```
    CASE 3:
      lr_door door_loc;
    CASE 4:
      rr_door door_loc;
    DEFAULT:
      EXIT;
  ENDTEST
```

Abhängig vom Wert des Registers `reg1` (1, 2, 3 oder 4) werden unterschiedliche Prozeduren aufgerufen, die die geeigneten Arbeiten an der gewählten Tür ausführen. Die Position der Tür wird im Argument `door_loc` angegeben.

Beispiel 2 - Dynamische Auswahl eines Prozeduraufrufs mit RAPID-Syntax

```
reg1 := 2;
%"proc"+NumToStr(reg1,0)% door_loc;
```

Die Prozedur `proc2` wird mit dem Argument `door_loc` aufgerufen.

Einschränkung: Alle Prozeduren müssen einen spezifischen Namen haben, z. B. `proc1, proc2, proc3..`

Beispiel 3 - Dynamische Auswahl eines Prozeduraufrufs mit CallByVar

```
reg1 := 2;
CallByVar "proc",reg1;
```

Die Prozedur `proc2` wird aufgerufen.

Einschränkung: Alle Prozeduren müssen einen spezifischen Namen haben, z. B. `proc1`, `proc2`, `proc3`, und es können keine Argumente verwendet werden.

Einschränkungen

Kann nur für den Aufruf von Prozeduren ohne Parameter verwendet werden.

Kann nicht zum Aufrufen von LOCAL-Prozeduren verwendet werden.

Die Abarbeitung von `CallByVar` beansprucht etwas mehr Zeit als die Abarbeitung eines normalen Prozeduraufrufs.

Syntax

```
CallByVar
  [Name ':='] <expression (IN) of string>','
  [Number ':='] <expression (IN) of num>';'
```

Weitere Informationen

Information über	Siehe
Aufruf von Prozeduren	*Technisches Referenzhandbuch - RAPID Overview* *Bedienungsanleitung - IRC5 mit FlexPendant*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.14 CamFlush – Entfernt die erfassten Daten für die Kamera.

Verwendung

`CamFlush` wird verwendet, um die erfassten `cameratarget` von der Kamera zu entfernen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamFlush`.

Beispiel 1:

```
CamFlush mycamera;
```

Die erfassten Daten für die Kamera `mycamera` wurden entfernt.

Argumente

```
CamFlush Camera
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Syntax

```
CamFlush
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.15 CamGetParameter – Ruft unterschiedlich benannte Kameraparameter ab.

Verwendung

`CamGetParameter` wird verwendet, um benannte Parameter abzurufen, die die Kamera verwendet. Der Benutzer muss den Namen des Parameters und seinen Rückgabetyp kennen, um seinen Wert abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamGetParameter`.

Beispiel 1:

```
VAR bool mybool:=FALSE;
...
CamGetParameter mycamera, "Pattern_1.Tool_Enabled_Status"
     \BoolVar:=mybool;
TPWite "The current value of Pattern_1.Tool_Enabled_Status is: "
     \Bool:=mybool;
```

Der benannte boolesche Parameter `Pattern_1.Tool_Enabled_Status` wird abgerufen, und der Wert wird am FlexPendant geschrieben.

Argumente

```
CamGetParameter Camera ParName [\NumVar] | [\BoolVar] | [\StrVar]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`ParName`

Parameter Name

Datentyp: `string`

Parametername in der Kamera

`[\NumVar]`

Datentyp: `num`

Variable (**VAR**) zum Speichern des numerischen Werts des abgerufenen Datenobjekts.

`[\BoolVar]`

Datentyp: `bool`

Variable (**VAR**) zum Speichern des booleschen Werts des abgerufenen Datenobjekts.

`[\StrVar]`

Datentyp: `string`

Variable (**VAR**) zum Speichern des Zeichenfolgenwerts des abgerufenen Datenobjekts.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Die Instruktion zeigt die spezifizierten Parameter direkt an, sobald die Instruktion ausgeführt wird und gibt den Wert zurück.

Wenn die Instruktion verwendet wird, um ein Ergebnis einer Bildanalyse anzuzeigen, stellen Sie sicher, dass die Kamera die Verarbeitung des Bilds beendet hat, bevor sie die Daten erhält.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CAM_BUSY	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
ERR_CAM_COM_TIMEOUT	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
ERR_CAM_GET_MISMATCH	Der mit der Instruktion CamGetParameter von der Kamera abgerufene Parameter hat den falschen Datentyp.
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden.

Syntax

```
CamGetParameter
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ ParName ':=' ] < expression (IN) of string >
  [ '\'NumVar ':=' < variable (VAR) of num > ]
  | [ '\'BoolVar ':=' < variable (VAR) of bool > ]
  | [ '\'StrVar ':=' < variable (VAR) of string > ] ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.16 CamGetResult – Ruft ein Kameraziel aus der Erfassung ab.

Verwendung

`CamGetResult` *(Ergebnisse der Kamera)* wird verwendet, um ein Kamera-Ergebnis vom Vision-Ergebnispuffer abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamGetResult`.

Beispiel 1:

```
VAR num mysceneid;
VAR cameratarget mycamtarget;
...
CamReqImage mycamera \SceneId:= mysceneid;
CamGetResult mycamera, mycamtarget \SceneId:= mysceneid;
```

Führen Sie den Befehl `mycamera` auf der Kamera aus, um ein Bild zu erfassen. Sie bekommen mit `SceneId` ein Vision-Ergebnis, das vom Bild stammt.

Argumente

```
CamGetResult Camera CamTarget [\SceneId] [\MaxTime]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`CamTarget`

Kameraziel

Datentyp: `cameratarget`

Die Variable, in der das Vision-Ergebnis gespeichert wird.

`[\SceneId]`

Szenenidentifikation

Datentyp: `num`

`SceneId` ist ein Bezeichner, der angibt, von welchem Bild das `cameratarget` erzeugt wurde.

`[\MaxTime]`

Maximale Wartezeit

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung auf eine Meldung wartet. Der maximal zulässige Wert ist 120 Sekunden.

Programmabarbeitung

`CamGetResult` ruft ein Kameraziel aus einer Vision-Ergebniserfassung ab. Wenn `SceneId` oder `MaxTime` nicht verwendet werden und es kein Ergebnis zum Abgerufen gibt, bleibt die Instruktion hängen. Wenn `SceneId` in `CamGetResult` verwendet wird, sollte es in einer vorherigen Instruktion `CamReqImage` erzeugt worden sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das `SceneId` kann nur verwendet werden, wenn das Bild von der Instruktion `CamReqImage` beauftragt wurde. Wenn Bilder von einem externen E/A-Signal generiert werden, kann das `SceneId` nicht in der Instruktion `CamGetResult` verwendet werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_MAXTIME`	Es konnten keine Ergebnisse während der Zeit des Timeouts abgerufen werden.
`ERR_CAM_NO_MORE_DATA`	Es können keine weiteren Vision-Ergebnisse für verwendete `SceneId` abgerufen werden, oder das Ergebnis konnte nicht während der Zeit des Timeouts abgerufen werden.

Syntax

```
CamGetResult
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ CamTarget ':=' ] < variable (VAR) of CameraTarget >
  [ '\'SceneId ':=' < expression (IN) of num > ]
  [ '\'MaxTime ':=' < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.17 CamLoadJob – Laden Sie eine Kameratask auf die Kamera

Verwendung

`CamLoadJob` *(Job Kamera laden)* lädt eine Kameratask, *job*, die Belichtungsparameter, Kalibrierung und die zu verwendenden Vision-Tools beschreibt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamLoadJob`.

Beispiel 1:

```
CamSetProgramMode mycamera;
CamLoadJob mycamera, "myjob.job";
CamSetRunMode mycamera;
```

Der Auftrag `myjob` wird auf die Kamera mit dem Namen `mycamera` geladen.

Argumente

```
CamLoadJob Camera JobName [\KeepTargets] [\MaxTime]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`Name`

Datentyp: `string`

Name des auf die Kamera zu ladenden Auftrags.

`[\KeepTargets]`

Datentyp: `switch`

Dieses Argument wird verwendet, um festzulegen, ob existierenden Kameraziele, die von der Kamera generiert werden, beibehalten werden sollten.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung auf eine Meldung wartet. Der maximal zulässige Wert ist 120 Sekunden.

Programmabarbeitung

Die Abarbeitung von `CamLoadJob` wartet, bis der Auftrag geladen ist, oder durch einen Timeoutfehler fehlschlägt. Wenn das optionale Argument `KeepTargets` verwendet wird, werden die alten erfassten Daten für die spezielle Kamera behalten. Das Standardverhalten ist die Entfernung der alten erfassten Daten (löschen).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CAM_BUSY	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
ERR_CAM_COM_TIMEOUT	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
ERR_CAM_MAXTIME	Der Kameraauftrag wurde nicht innerhalb der Timeoutzeit geladen.
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden
ERR_CAM_NO_PROGMODE	Die Kamera befindet sich nicht im Programmmodus.

Einschränkungen

CamLoadJob kann nur ausgeführt werden, wenn die Kamera im Programmmodus ist. Verwenden Sie die Instruktion CamSetProgramMode, um die Kamera in den Programmmodus zu bringen.

Um den Auftrag laden zu können, muss die Auftragsdatei auf der Flashdisk der Kamera gespeichert sein.

Syntax

```
CamLoadJob
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ JobName ':=' ] <expression (IN) of string >
  [ '\'KeepTargets ]
  [ '\'MaxTime ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.18 CamReqImage – Senden Sie einen Befehl an die Kamera, ein Bild zu erfassen.

Verwendung

`CamReqImage` *(Kamera Bildanforderung)* weist die Kamera an, ein Bild zu erfassen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamReqImage`.

Beispiel 1:

```
CamReqImage mycamera;
```

Weisen Sie die Kamera `mycamera` an, ein Bild zu erfassen.

Argumente

```
CamReqImage Camera [\SceneId] [\KeepTargets] [\AwaitComplete]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`[\SceneId]`

Szenenidentifikation

Datentyp: `num`

Das optionale Argument `SceneId` ist ein Bezeichner für das erfasste Bild. Es wird für jedes ausgeführte `CamReqImage` generiert, das das optionale Argument `SceneId`verwendet. Der Bezeichner ist eine ganze Zahl zwischen 1 und 8388608. Wenn kein `SceneId` verwendet wird, wird der Bezeichnerwert auf 0 gesetzt.

`[\KeepTargets]`

Datentyp: `switch`

Dieses Argument wird verwendet, um festzulegen, ob alte erfasste Daten für eine spezifische Kamera behalten werden sollten.

`[\AwaitComplete]`

Datentyp: `switch`

Wenn das optionale Argument `\AwaitComplete` angegeben ist, wartet die Instruktion, bis die Ergebnisse von dem Bild empfangen wurden.

Wenn `\AwaitComplete` verwendet wird, muss der Kameraauslösertyp auf **Extern** eingestellt sein.

Programmabarbeitung

`CamReqImage`weist eine spezifische Kamera an, ein Bild zu erfassen. Wenn das optionale Argument `SceneId` verwendet wird, werden die verfügbaren optischen Ergebnisse eines erhaltenen Bilds mit der eindeutigen Nummer, die von der Instruktion generiert wird, markiert.

Wenn das optionale Argument `KeepTargets` verwendet wird, werden die alten erfassten Daten der spezifischen Kamera behalten. Das Standardverhalten ist die Entfernung aller alten erfassten Daten (löschen).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CAM_BUSY	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
ERR_CAM_COM_TIMEOUT	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
ERR_CAM_NO_RUNMODE	Die Kamera befindet sich nicht im Abarbeitungsmodus.
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden.

Einschränkungen

CamReqImage kann nur ausgeführt werden, wenn die Kamera im Abarbeitungsmodus ist. Verwenden Sie die Instruktion CamSetRunMode, um die Kamera in den Abarbeitungsmodus zu bringen.

Syntax

```
CamReqImage
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ '\'SceneId ':=' < variable (VAR) of num > ]
  [ '\'KeepTargets ]
  [ '\'AwaitComplete ]';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.19 CamSetExposure – Legen Sie die spezifischen Daten der Kamera fest

Verwendung

`CamSetExposure`(*Kamera Belichtungseinstellung*) legt die spezifischen Daten der Kamera fest und ermöglicht, abhängig von den Lichtverhältnissen der Umgebung, die Anpassung der Bildparameter.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamSetExposure`.

Beispiel 1:

```
CamSetExposure mycamera \ExposureTime:=10;
```

Weisen Sie die Kamera `mycamera` an, die Belichtungszeit auf 10 ms zu ändern.

Argumente

```
CamSetExposure Camera [\ExposureTime] [\Brightness] [\Contrast]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`[\ExposureTime]`

Datentyp: `num`

Wenn dieses optionale Argument verwendet wird, wird die Belichtungszeit der Kamera aktualisiert. Der Wert wird in Millisekunden (ms) angegeben.

`[\Brightness]`

Datentyp: `num`

Wenn dieses optionale Argument verwendet wird, wird die Helligkeitseinstellung der Kamera aktualisiert. Der Wert wird normalerweise auf einer Skala von 0 bis 1 angegeben.

`[\Contrast]`

Datentyp: `num`

Wenn dieses optionale Argument verwendet wird, wird die Kontrasteinstellung der Kamera aktualisiert. Der Wert wird normalerweise auf einer Skala von 0 bis 1 angegeben.

Programmabarbeitung

Die Instruktion aktualisiert die Belichtungszeit, Helligkeit und den Kontrast, wenn es möglich ist, diese für die spezifische Kamera zu aktualisieren. Wenn eine Einstellung von einer Kamera nicht unterstützt wird, wird eine Fehlermeldung für den Benutzer angezeigt, und die Programmausführung wird gestoppt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CAM_COM_TIMEOUT	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden.

Syntax

```
CamSetExposure
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ '\'ExposureTime ':=' < variable (IN) of num > ]
  [ '\'Brightness ':=' < variable (IN) of num > ]
  [ '\'Contrast ':=' < variable (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.20 CamSetParameter – Legen Sie unterschiedlich benannte Kameraparameter fest.

Verwendung

`CamSetParameter` wird verwendet, um unterschiedlich benannte Kameraparameter festzulegen, die eine Kamera aufweisen kann. Mit dieser Instruktion ist es möglich, unterschiedliche Parameter in der Kamera während der Laufzeit zu ändern. Der Benutzer muss den Namen des Parameters und seinen Typ kennen, um seinen Wert einzustellen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamSetParameter`.

Beispiel 1:

```
CamSetParameter mycamera, "Pattern_1.Tool_Enabled" \BoolVal:=FALSE;
CamSetRunMode mycamera;
```

In diesem Beispiel ist der Parameter mit dem Namen „`Pattern_1.Tool_Enabled`“ auf falsch eingestellt, was bedeutet, dass das spezifische Vision-Tool nicht ausgeführt werden soll, wenn ein Bild erfasst wird.

Dadurch wird das Visionwerkzeug schneller abgearbeitet. Allerdings produziert das deaktivierte Werkzeug immer noch Ergebnisse mit den Werten aus der letzten aktiven Abarbeitung. Damit diese Ziele nicht benutzt werden, sind sie im RAPID-Programm auszusortieren.

Argumente

```
CamSetParameter Camera ParName [\NumVal] | [\BoolVal] | [\StrVal]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`ParName`

Datentyp: `string`

Parametername in der Kamera

`[\NumVal]`

Datentyp: `num`

Der numerische Wert, auf den der Kameraparameter festgelegt werden muss, mit dem Namen, der im Argument`ParName` festgelegt ist.

`[\BoolVal]`

Datentyp: `bool`

Der boolesche Wert, auf den der Kameraparameter festgelegt werden muss, mit dem Namen, der im Argument`ParName` festgelegt ist.

`[\StrVal]`

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Wert der Zeichenfolge, auf den der Kameraparameter festgelegt werden muss, mit dem Namen, der im Argument`ParName` festgelegt ist.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_COM_TIMEOUT`	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden.
`ERR_CAM_SET_MISMATCH`	Der auf die Kamera mit der Instruktion CamSetParameter geschriebene Parameter hat den falschen Datentyp oder der Wert liegt außerhalb des zulässigen Bereichs.

Einschränkungen

Vom Benutzer erstellte Parameter sind ausschließlich vom Typ `EditString`, `EditInt` oder `EditFloat`.

Syntax

```
CamSetParameter
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ ParName ':=' ] < expression (IN) of string >
  [ '\'NumVal ':=' < expression (IN) of num > ]
  | [ '\'BoolVal ':=' < expression (IN) of bool > ]
  | [ '\'StrVal ':=' < expression (IN) of string > ] ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.21 CamSetProgramMode – Weist die Kamera an, in den Programmmodus zu wechseln

Verwendung

`CamSetProgramMode` *(Kamera Einstellung Programmmodus)* weist die Kamera an, in den Programmmodus (offline) zu wechseln.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamSetProgramMode`.

Beispiel 1:

```
CamSetProgramMode mycamera;
CamLoadJob mycamera, "myjob.job";
CamSetRunMode mycamera;
...
```

Bringen Sie zunächst die Kamera in den Programmmodus. Laden Sie dann`myjob` auf die Kamera. Weisen Sie anschließend die Kamera an, in den Abarbeitungsmodus zu wechseln.

Argumente

```
CamSetProgramMode Camera
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Programmabarbeitung

Wenn der Kamera mit der Instruktion`CamSetProgramMode` angewiesen wird, in den Programmmodus zu wechseln, wird es möglich, die Einstellungen zu ändern und einen Auftrag auf die Kamera zu laden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_COM_TIMEOUT`	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden.

Syntax

```
CamSetProgramMode
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.22 CamSetRunMode – Weist die Kamera an, in den Abarbeitungsmodus zu wechseln

Verwendung

`CamSetRunMode` *(Kamera Einstellung Abarbeitungsmodus)* weist die Kamera an, in den Abarbeitungsmodus (online) zu wechseln, und aktualisiert die Steuerung am aktuellen Ausgang auf RAPID-Konfiguration.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamSetRunMode`.

Beispiel 1:

```
CamSetProgramMode mycamera;
CamLoadJob mycamera, "myjob.job";
...
CamSetRunMode mycamera;
```

Bringen Sie zunächst die Kamera in den Programmmodus. Laden Sie dann `myjob` auf die Kamera. Weisen Sie anschließend die Kamera mit der Instruktion `CamSetRunMode` an, in den Abarbeitungsmodus zu wechseln.

Argumente

```
CamSetRunMode Camera
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Programmabarbeitung

Wenn eine Kamera mit `CamSetRunMode` angewiesen wird, in den Abarbeitungsmodus zu wechseln, ist es möglich, Bilder aufzunehmen.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_COM_TIMEOUT`	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden.

Syntax

```
CamSetRunMode
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.23 CamStartLoadJob – Starten Sie damit, eine Kameratask auf die Kamera zu laden

Verwendung

`CamStartLoadJob` beginnt damit, einen Job auf eine Kamera zu laden, und anschließend fährt die Abarbeitung mit der nächsten Instruktion fort. Während des Ladens können andere Instruktionen parallel ausgeführt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamStartLoadJob`.

Beispiel 1:

```
...
CamStartLoadJob mycamera, "myjob.job";
MoveL p1, v1000, fine, tool2;
CamWaitLoadJob mycamera;
CamSetRunMode mycamera;
CamReqImage mycamera;
...
```

Zunächst wird die Ladung eines Auftrags auf die Kamera gestartet, und während des Ladens erfolgt eine Bewegung zu Position P1. Wenn die Bewegung abgeschlossen ist und das Laden beendet ist, wird ein Bild erfasst.

Argumente

```
CamStartLoadJob Camera Name [\KeepTargets]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`Name`

Datentyp: `string`

Name des auf die Kamera zu ladenden Auftrags.

`[\KeepTargets]`

Datentyp: `switch`

Dieses Argument wird verwendet, um festzulegen, ob alte erfasste Daten für eine spezifische Kamera behalten werden sollten.

Programmabarbeitung

Die Abarbeitung von `CamStartLoadJob` fordert den Ladevorgang nur an und fährt dann direkt mit der nächsten Instruktion fort, ohne den Abschluss des Ladevorgangs abzuwarten. Wenn das optionale Argument `\KeepTargets` verwendet wird, werden die alten erfassten Daten für die spezifische Kamera nicht entfernt. Das Standardverhalten ist das Löschen der alten erfassten Daten (löschen).

Die Kamera wird den Lastvorgang überwachen und keine neuen Kameraanforderungen akzeptieren, bevor der Vorgang mit `CamWaitLoadJob`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

beendet ist. Allerdings mit der Ausnahme, dass `CamStartSetParameter`-Anfragen in die Warteschlange gestellt werden können.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden

Einschränkungen

`CamStartLoadJob` kann nur ausgeführt werden, wenn die Kamera im Programmmodus ist. Verwenden Sie die Instruktion `CamSetProgramMode`, um die Kamera in den Programmmodus zu bringen.

Wenn eine laufenden Ladung eines Auftrags ausgeführt wird, kann auf die spezifische Kamera nicht mit einer anderen Instruktion oder Funktion zugegriffen werden. Die darauffolgende Kamerainstruktion oder -funktion muss eine `CamWaitLoadJob`-Instruktion sein.

Um den Auftrag laden zu können, muss die Auftragsdatei auf der Flashdisk der Kamera gespeichert sein.

Syntax

```
CamStartLoadJob
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ Name ':='] <expression (IN) of string >
  [ '\'KeepTargets ]';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*
`CamWaitLoadJob`	*CamWaitLoadJob – Warten Sie, bis eine Kameratask geladen wird auf Seite 77*
`CamStartSetParameter`	*CamStartSetParameter – Start des Setzen-Vorgangs einer Parametereinstellung. auf Seite 74*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.24 CamStartSetParameter – Start des Setzen-Vorgangs einer Parametereinstellung.

Verwendung

`CamStartSetParameter` wird verwendet, um den Setzen-Vorgang einer Parametereinstellung in der Kamera zu beginnen. Während der Setzen-Vorgang ausgeführt wird, können andere RAPID-Instruktionen und Funktionen parallel dazu ausgeführt werden. Die Kamera wird den Vorgang „Parameter setzen“ ausführen und keine neuen Anforderungen akzeptieren, bevor dieser Vorgang mit `CamWaitSetParameter` abgeschlossen ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `CamStartSetParameter`.

Beispiel 1:

```
CamStartSetParameter mycamera, "Pattern_1.Tool_Enabled"
     \BoolVal:=FALSE;
MoveL p1, v1000, fine, tool2;
CamWaitSetParameter mycamera;
```

Zuerst wird das Setzen eines Parameters angefordert und während der Ausführung des Setzen-Vorgangs wird eine Bewegung zur Position p1 ausgeführt. Wenn die Bewegung bereit ist und wenn das Setzen des Parameters bereit ist, dann wird die RAPID-Abarbeitung fortgesetzt.

Beispiel 2:

```
CamStartSetParameter mycamera,
     ”Pattern_1.Description”\StrVal:=”mydescription”;
CamStartSetParameter mycamera,
     ”Pattern_1.Rotation_Tolerance”\NumVal:=15;
MoveL p1, v1000, fine, tool2;
CamWaitSetParameter mycamera;
CamWaitSetParameter mycamera;
```

Es werden zwei Parametereinstellungen angefordert. Während diese abgearbeitet werden, wird eine Bewegung ausgeführt. Die beiden `CamStartSetParameter`-Instruktionen müssen mit den beiden `CamWaitSetParameter`-Instruktionen abgeglichen werden, damit die andere Anforderung für die Kamera ausgeführt werden kann.

Argumente

```
CamStartSetParameter Camera ParName [\NumVal] | [\BoolVal] |
     [\StrVal]
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`ParName`

Datentyp: `string`

Parametername in der Kamera

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\NumVal]`

Datentyp: `num`

Der numerische Wert, auf den der Kameraparameter festgelegt werden muss, mit dem Namen, der im Argument`ParName` festgelegt ist.

`[\BoolVal]`

Datentyp: `bool`

Der boolesche Wert, auf den der Kameraparameter festgelegt werden muss, mit dem Namen, der im Argument`ParName` festgelegt ist.

`[\StrVal]`

Datentyp: `string`

Der Wert der Zeichenfolge, auf den der Kameraparameter festgelegt werden muss, mit dem Namen, der im Argument`ParName` festgelegt ist.

Programmabarbeitung

`CamStartSetParameter` startet den Setzen-Vorgang eines Parameters in der Kamera. Während der Setzen-Vorgang ausgeführt wird, werden parallel dazu andere RAPID-Instruktionen und Funktionen ausgeführt.

Die Kamera wird den Parameter-setzen-Vorgangs ausführen und sie wird andere Anforderungen erst dann ausführen, wenn dieser Vorgang mit `CamWaitSetParameter` abgeschlossen ist. Jede `CamStartSetParameter`-Instruktion muss mit einer entsprechenden Instruktion `CamWaitSetParameter` abgeglichen werden.

Bei mehreren `StartSetParameter`-Anforderungen in der Warteschlange spiegelt der entsprechende `WaitSetParameter` die identische Reihenfolge der Anforderung wider und gibt somit den Status für diesen `StartSetParameter` aus.

Hinweis

Vom Anwender definierte Parameter in der Kamera müssen mit einer der folgenden Kamerafunktionen verknüpft werden:

- `EditFloat`
- `EditInt`
- `EditString`

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden.
`ERR_CAM_NO_PROGMODE`	Die Kamera befindet sich nicht im Programmmodus.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.24 CamStartSetParameter – Start des Setzen-Vorgangs einer Parametereinstellung.

RobotWare - OS

Fortsetzung

Einschränkungen

Die Steuerung kann 10 ausstehende `CamStartSetParameter` Operationen haben. Wenn die Steuerung 10 ausstehenden Anforderungen hat, müssen alle Anforderungen mit `CamWaitSetParameter`-Instruktionen bestätigt werden, bevor ein neuer `CamStartSetParameter` angefordert wird.

Syntax

```
CamStartSetParameter
  [ Camera ':=' ] < variable (VAR) of cameradev > ','
  [ ParName ':=' ] < expression (IN) of string >
  [ '\'NumVal ':=' < expression (IN) of num > ]
  | [ '\'BoolVal ':=' < expression (IN) of bool > ]
  | [ '\'StrVal ':=' < expression (IN) of string > ] ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*
`CamWaitSetParameter`	*CamWaitSetParameter – Warten, bis ein neuer Set-Vorgang bereit ist. auf Seite 79*
`CamSetParameter`	*CamSetParameter – Legen Sie unterschiedlich benannte Kameraparameter fest. auf Seite 66*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.25 CamWaitLoadJob – Warten Sie, bis eine Kameratask geladen wird

Verwendung

`CamWaitLoadJob` *(Kamera Warten Job laden)* wartet, bis das Laden eines Jobs auf die Kamera beendet ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamWaitLoadJob`.

Beispiel 1:

```
...
CamStartLoadJob mycamera, "myjob.job";
MoveL p1, v1000, fine, tool2;
CamWaitLoadJob mycamera;
CamSetRunMode mycamera;
CamReqImage mycamera;
...
```

Zunächst wird die Ladung eines Auftrags auf die Kamera gestartet, und während des Ladens erfolgt eine Bewegung zu Position P1. Wenn die Bewegung abgeschlossen ist und das Laden beendet ist, wird ein Bild erfasst.

Argumente

```
CamWaitLoadJob Camera
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_COM_TIMEOUT`	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden

Einschränkungen

`CamWaitLoadJob` kann nur ausgeführt werden, wenn die Kamera im Programmmodus ist. Verwenden Sie die Instruktion `CamSetProgramMode`, um die Kamera in den Programmmodus zu bringen.

Wenn eine laufenden Ladung eines Auftrags ausgeführt wird, kann auf die spezifische Kamera nicht mit einer anderen Instruktion oder Funktion zugegriffen werden. Die darauffolgende Kamerainstruktion oder -funktion muss eine `CamWaitLoadJob`-Instruktion sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.25 CamWaitLoadJob – Warten Sie, bis eine Kameratask geladen wird

Integrated Vision

Fortsetzung

Syntax

```
CamWaitLoadJob
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*
`CamStartLoadJob`	*CamStartLoadJob – Starten Sie damit, eine Kameratask auf die Kamera zu laden auf Seite 72*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.26 CamWaitSetParameter – Warten, bis ein neuer Set-Vorgang bereit ist.

Verwendung

`CamWaitSetParameter` wartet, bis der Setzen-Vorgang des Parameters in der Kamera ausgeführt wurde und den Status zurückgibt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CamWaitSetParameter`.

Beispiel 1:

```
CamStartSetParameter mycamera, "Pattern_1.Tool_Enabled"
     \BoolVal:=FALSE;
MoveL p1, v1000, fine, tool2;
CamWaitSetParameter mycamera;
```

Zuerst wird das Setzen eines Parameters angefordert und während der Ausführung des Setzen-Vorgangs wird eine Bewegung zur Position p1 ausgeführt. Wenn die Bewegung bereit ist und wenn das Setzen des Parameters bereit ist, dann wird die RAPID-Abarbeitung fortgesetzt.

Argumente

```
CamWaitSetParameter Camera
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_COM_TIMEOUT`	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
`ERR_CAM_NOT_ON_NETWORK`	Die Kamera ist nicht verbunden.
`ERR_CAM_NO_START_SET_PARAMETER`	Keine kontinuierliche Anforderung für das Setzen eines Parameters für die Kamera.
`ERR_CAM_SET_MISMATCH`	Der auf die Kamera mit der Instruktion `CamSetParameter` geschriebene Parameter hat den falschen Datentyp oder der Wert liegt außerhalb des zulässigen Bereichs.
`ERR_CAM_SET_PARAMETER_REJECTED`	Die vorhergehende Anweisung ist fehlgeschlagen und Parameter setzen ist ebenfalls fehlgeschlagen für Kamera.

Syntax

```
CamWaitSetParameter
  [ Camera ':=' ] < variable (VAR) of cameradev > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.26 CamWaitSetParameter – Warten, bis ein neuer Set-Vorgang bereit ist.

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*
`CamStartSetParameter`	*CamStartSetParameter – Start des Setzen-Vorgangs einer Parametereinstellung. auf Seite 74*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.27 CancelLoad - Abbruch des Ladens eines Moduls

Verwendung

`CancelLoad` kann zum Abbrechen des mit der Instruktion `StartLoad`. generierten Ladevorgangs verwendet werden.

`CancelLoad` kann nur zwischen der Instruktion `StartLoad` und `WaitLoad` verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CancelLoad`:

Siehe auch *Weitere Beispiele auf Seite 81*.

Beispiel 1

```
CancelLoad load1;
```

Der Ladevorgang `load1` wird abgebrochen.

Argumente

```
CancelLoad LoadNo
```

`LoadNo`

Datentyp: `loadsession`

Verweis auf den Ladevorgang, erstellt durch die Instruktion `StartLoad`.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_LOADNO_NOUSE`	Die im Argument `LoadNo` angegebene Variable wird nicht verwendet, d. h., dass kein Ladevorgang abläuft.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `CancelLoad` sind unten aufgeführt.

Beispiel 1:

```
VAR loadsession load1;

StartLoad "HOME:"\File:="PART_B.MOD",load1;
...
IF ...
  CancelLoad load1;
  StartLoad "HOME:"\File:="PART_C.MOD",load1;
ENDIF
...
WaitLoad load1;
```

Die Instruktion `CancelLoad` bricht den laufenden Ladevorgang des Moduls `PART_B.MOD` ab und ermöglicht, stattdessen `PART_C.MOD` zu laden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.27 CancelLoad - Abbruch des Ladens eines Moduls

RobotWare - OS

Fortsetzung

Einschränkung

`CancelLoad` kann nur verwendet werden, nachdem die Instruktion `StartLoad` abgearbeitet wurde und bevor die Instruktion `WaitLoad` gestartet wurde.

Syntax

```
CancelLoad
  [ LoadNo ':=' ] < variable (VAR) of loadsession >';'
```

Weitere Informationen

Information über	Siehe
Laden eines Programmmoduls während der Abarbeitung	*StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802*
Verbinden des geladenen Moduls mit der Task	*WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*
Ladevorgang	*loadsession – Ladevorgang für ein Programm auf Seite 1730*
Laden eines Programmmoduls	*Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346*
Entfernen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*
Überprüft Programmreferenzen.	*CheckProgRef - Überprüfen von Programmverweisen auf Seite 131*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.28 CapAPTrSetup - Einrichtung eines At-Point-Trackers

Verwendung

`CapAPTrSetup` *(Einrichtung eines At-Point-Trackers)*, um einen At-Point-Tracker-Sensortyp einzurichten, beispielsweise *WeldGuide* oder AWC.

Die Sensorschnittstelle kommuniziert mithilfe des RTP1-Transportprotokolls mit höchstens einem Sensor über serielle Kanäle.

Grundlegendes Beispiel

SIO.cfg:

```
COM_TRP:
-Name "wg:" -Type "SOCKETDEV" -RemoteAddress "192.168.1255.101"
     -RemotePort "6344"
```

RAPID-Code:

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
VAR pos SensorPos;

! Setup a Weldguide
CapAPTrSetup "wg:", do_left, 80, do_right, 80;
```

Argumente

```
CapAPTrSetup device DoLeft LevelLeft DoRight LevelRight [\LogFile]
     [\LogSize]
```

device

Datentyp: `string`

Die Konfiguration des E/A-Gerätenamens ist in der Datei sio.cfg definiert.

DoLeft

Datentyp: `signaldo`

Das digitale Ausgangssignal für die Synchronisation der Pendelbewegung auf der linken Seite des Pendelzyklus.

LevelLeft

Datentyp: `num`

Die Koordinierungsposition auf der linken Seite des Pendelbewegungsmusters. Der Wert wird als Prozentsatz der Breite links von der Pendelmitte angegeben. Wenn die Pendelbewegung über diesen Punkt hinausgeht, wird automatisch ein hohes digitales Ausgangssignal festgelegt (wenn das Signal definiert ist).

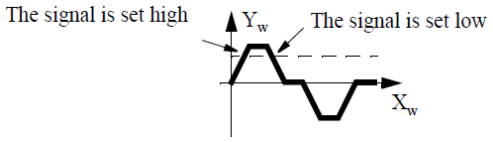
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.28 CapAPTrSetup - Einrichtung eines At-Point-Trackers

Continuous Application Platform (CAP)
Fortsetzung

Die Koordinationsart ist für die Nahtverfolgung mittels des Through-the-Arc-Trackers gedacht.



xx1200000178

DoRight

Datentyp: `signaldo`

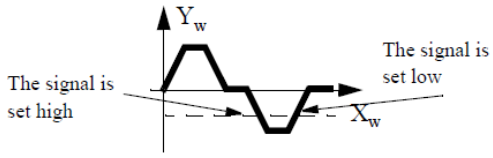
Das digitale Ausgangssignal für die Synchronisation der Pendelbewegung auf der rechten Seite des Pendelzyklus.

LevelRight

Datentyp: `num`

Die Koordinierungsposition auf der rechten Seite des Pendelbewegungsmusters. Der Wert wird als Prozentsatz der Breite rechts der Pendelmitte angegeben. Wenn die Pendelbewegung über diesen Punkt hinausgeht, wird automatisch ein hohes digitales Ausgangssignal festgelegt (wenn das Signal definiert ist).

Die Koordinationsart ist für die Nahtverfolgung mittels des Through-the-Arc-Trackers gedacht.



xx1200000179

[\LogFile]

Datentyp: `string`

Name der Tracklog-Protokolldatei

[\LogSize]

Datentyp: `num`

Größe des Tracklog-Ringpuffers, das heißt die Anzahl der Sensormessungen, die während der Verfolgung gepuffert werden können.

Standardwert: 1000.

Syntax

```
CapAPTrSetup
  [device ':='] < expression (IN) of string> ','
  [DoLeft ':='] < expression (IN) of signaldo > ','
  [LevelLeft ':='] < expression (IN) of num > ','
  [DoRight ':='] < expression (IN) of signaldo > ','
  [LevelRight ':='] < expression (IN) of num >
  ['\'LogFile ':='] < expression (IN) of string >
  ['\'LogSize ':='] < expression (IN) of num > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

	Beschrieben in:
Continuous Application Platform	*Application manual - Continuous Application Platform*
Sensor Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.29 CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird.

Verwendung

`CapAPTrSetupAI` wird verwendet, um einen At-Point-Tracker einzurichten, der von analogen Eingangssignalen gesteuert wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CapAPTrSetupAI`.

Beispiel 1:

```
TASK PERS capdata cData:=[.....];
TASK PERS weavestartdata wsData:=[.....];
TASK PERS capweavedata wData:=[.....];
TASK PERS captrackdata trackData:=["ANALOG_TRACKER",.....];

VAR capaptrreferencedata referenceData:=[2,2,1,1,0.1,0.1];
VAR signalai ai_y;
VAR signalai ai_z;

AliasIO realsignal_y, ai_y;
AliasIO realsignal_z, ai_z;
CapAPTrSetupAI ai_y, ai_z, referenceData;

CapL p1, v200, cData, wsData, wData , fine, tWeldGun
     \Track:=trackData;
```

Argumente

```
CapAPTrSetupAO ai_y, ai_z, ReferenceData [\MaxIncrCorr]
     [\WarnMaxCorr] [\Filter] [\SampleTime] [\Logfile] [\LogSize]
     [\LatestCorr] [\AccCorr]
```

`ai_y`

Datentyp: `signalai`

Analoges Eingangssignal, das als Prozessposition für die y-Richtung verwendet wird.

`ai_z`

Datentyp: `signalai`

Analoges Eingangssignal, das als Prozessposition für die z-Richtung verwendet wird.

`ReferenceData`

Datentyp: `capaptrreferencedata`

Für die Korrekturreglerschleife verwendete Setup-Daten.

`MaxIncCorr`

Datentyp: `num`

Zulässige maximale inkrementelle Korrektur (in mm).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.29 CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird.

Continuous Application Platform (CAP)

Fortsetzung

Wenn die inkrementelle TCP-Korrektur größer als `\MaxIncCorr` und `\WarnMaxCorr` ist, setzt der Roboter zwar seine Bahn fort, die übernommene inkrementelle Korrektur überschreitet jedoch nicht `\MaxIncCorr`. Wenn `\WarnMaxCorr` nicht angegeben ist, wird ein Verfolgungsfehler gemeldet und die Programmausführung angehalten.

`WarnMaxCorr`

Datentyp: `switch`

Wenn dieser Schalter vorhanden ist, wird die Programmausführung nicht unterbrochen, wenn der Grenzwert für die maximale Korrektur überschritten wird, angegeben in `\MaxIncCorr`. Es wird lediglich eine Warnung ausgegeben.

`Filter`

Datentyp: `num`

Größe des Referenz-Beispieldatenfilters. Ein Wert zwischen 1 und 15 ist erlaubt; der Standardwert ist 1.

`SampleTime`

Datentyp: `num`

Abtastrate in Millisekunden für die Korrekturschleife. Der Wert wird auf ein Vielfaches von 24 gerundet. Der zulässige Mindestwert ist 24 und der Standardwert ist 24.

`LogFile`

Datentyp: `string`

Der Name der tracklog-Protokolldatei. Die Protokolldatei wird im HOME -Verzeichnis des Systems abgelegt.

`LatestCorr`

Datentyp: `pos`

Größe der zuletzt hinzugefügten Korrektur (in mm).

`AccCorr`

Datentyp: `pos`

Größe der insgesamt akkumulierten Korrektur (in mm) hinzugefügt.

`LogSize`

Datentyp: `num`

Größe des tracklog-Ringpuffers, d. h. die Anzahl der Sensormessungen, die während der Verfolgung gepuffert werden können.

Standardwert: 1000.

Syntax

```
CapAPTrSetupAI
  [aoi_y ':='] <expression (IN) of signalai> ','
  [ai_z ':='] <expression (IN) of signalai> ','
  [ReferenceData ':='] <expression (IN) of capaptrreferencedata>
       ','
  [\MaxIncrCorr ':='] <expression (IN) of num> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.29 CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird.

Continuous Application Platform (CAP)

Fortsetzung

```
[\WarnMaxCorr ':='] <expression (IN) of switch> ','
[\Filter ':='] <expression (IN) of num> ','
[\SampleTime ':='] <expression (IN) of num> ','
[\LogFile ':='] <expression (IN) of string> ','
[\LogSize ':='] <expression (IN) of num> ','
[\LatestCorr ':='] <expression (PERS) of pos> ','
[\AccCorr ':='] <expression (PERS) of pos> ';'
```

Weitere Informationen

Information über	Siehe
Instruktion `CapAPTrSetupAO`	*CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird. auf Seite 89*
Instruktion `CapAPTrSetupPERS`	*CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird. auf Seite 92*
Datentyp `capaptrreferencedata`	*capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker auf Seite 1651*
Continuous Application Platform	*Application manual - Continuous Application Platform*
Sensor Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.30 CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird.

Verwendung

`CapAPTrSetupAO` wird verwendet, um einen At-Point-Tracker einzurichten, der von analogen Ausgangssignalen gesteuert wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CapAPTrSetupAO`.

Beispiel 1:

```
TASK PERS capdata cData:=[.....];
TASK PERS weavestartdata wsData:=[.....];
TASK PERS capweavedata wData:=[.....];
TASK PERS captrackdata trackData:=["ANALOG_TRACKER",.....];

VAR capaptrreferencedata referenceData:=[2,2,1,1,0.1,0.1];
VAR signalao ao_y;
VAR signalao ao_z;

AliasIO realsignal_y, ao_y;
AliasIO realsignal_z, ao_z;
CapAPTrSetupAO ao_y, ao_z, referenceData;

CapL p1, v200, cData, wsData, wData , fine, tWeldGun
     \Track:=trackData;
```

Argumente

```
CapAPTrSetupAO ao_y, ao_z, ReferenceData [\MaxIncrCorr]
     [\WarnMaxCorr] [\Filter] [\SampleTime] [\Logfile] [\LogSize]
     [\LatestCorr] [\AccCorr]
```

`ao_y`

Datentyp: `signalao`

Analoges Ausgangssignal, das als Prozessposition für die y-Richtung verwendet wird.

`ao_z`

Datentyp: `signalao`

Analoges Ausgangssignal, das als Prozessposition für die z-Richtung verwendet wird.

`ReferenceData`

Datentyp: `capaptrreferencedata`

Für die Korrekturreglerschleife verwendete Setup-Daten.

`MaxIncCorr`

Datentyp: `num`

Zulässige maximale inkrementelle Korrektur (in mm).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.30 CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird.

Continuous Application Platform (CAP)

Fortsetzung

Wenn die inkrementelle TCP-Korrektur größer als `\MaxIncCorr` und `\WarnMaxCorr` ist, setzt der Roboter zwar seine Bahn fort, die übernommene inkrementelle Korrektur überschreitet jedoch nicht `\MaxIncCorr`. Wenn `\WarnMaxCorr` nicht angegeben ist, wird ein Verfolgungsfehler gemeldet und die Programmausführung angehalten.

`WarnMaxCorr`

Datentyp: `switch`

Wenn dieser Schalter vorhanden ist, wird die Programmausführung nicht unterbrochen, wenn der Grenzwert für die maximale Korrektur überschritten wird, angegeben in `\MaxIncCorr`. Es wird lediglich eine Warnung ausgegeben.

`Filter`

Datentyp: `num`

Größe des Referenz-Beispieldatenfilters. Ein Wert zwischen 1 und 15 ist erlaubt; der Standardwert ist 1.

`SampleTime`

Datentyp: `num`

Abtastrate in Millisekunden für die Korrekturschleife. Der Wert wird auf ein Vielfaches von 24 gerundet. Der zulässige Mindestwert ist 24 und der Standardwert ist 24.

`LogFile`

Datentyp: `string`

Der Name der tracklog-Protokolldatei. Die Protokolldatei wird im HOME -Verzeichnis des Systems abgelegt.

`LogSize`

Datentyp: `num`

Größe des tracklog-Ringpuffers, d. h. die Anzahl der Sensormessungen, die während der Verfolgung gepuffert werden können.

Standardwert: 1000.

`LatestCorr`

Datentyp: `pos`

Größe der zuletzt hinzugefügten Korrektur (in mm).

`AccCorr`

Datentyp: `pos`

Größe der insgesamt akkumulierten Korrektur (in mm) hinzugefügt.

Syntax

```
CapAPTrSetupAO
  [ao_y ':='] <expression (IN) of signalao> ','
  [ao_z ':='] <expression (IN) of signalao> ','
  [ReferenceData ':='] <expression (IN) of capaptrreferencedata>
       ','
  [\MaxIncrCorr ':='] <expression (IN) of num> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.30 CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird.

Continuous Application Platform (CAP)

Fortsetzung

```
[\WarnMaxCorr ':='] <expression (IN) of switch> ','
[\Filter ':='] <expression (IN) of num> ','
[\SampleTime ':='] <expression (IN) of num> ','
[\LogFile ':='] <expression (IN) of string> ','
[\LogSize ':='] <expression (IN) of num> ','
[\LatestCorr ':='] <expression (PERS) of pos> ','
[\AccCorr ':='] <expression (PERS) of pos> ';'
```

Weitere Informationen

Information über	Siehe
Instruktion `CapAPTrSetupAI`	*CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird. auf Seite 86*
Instruktion `CapAPTrSetupPERS`	*CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird. auf Seite 92*
Datentyp `capaptrreferencedata`	*capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker auf Seite 1651*
Continuous Application Platform	*Application manual - Continuous Application Platform*
Sensor Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.31 CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird.

Verwendung

`CapAPTrSetupPERS` wird verwendet, um einen At-Point-Tracker einzurichten, der von persistenten Variablen gesteuert wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CapAPTrSetupPERS`.

Beispiel 1:

```
TASK PERS capdata cData:=[.....];
TASK PERS weavestartdata wsData:=[.....];
TASK PERS capweavedata wData:=[.....];
TASK PERS captrackdata trackData:=["ANALOG_TRACKER",.....];
PERS pos corr:=[0,-0.05,-0.025];

VAR capaptrreferencedata referenceData:=[2,2,1,1,0.1,0.1];

IDelete intno1;
CONNECT intno1 WITH trOffset;
CapAPTRSetupPERS corr.y, corr.z, referenceData;

ITimer 1,intno1;
CapL p1, v200, cData, wsData, wData , fine,
     tWeldGun\Track:=trackData;

TRAP trOffset
  corr.y := referenceData.reference_y +- .....;
  corr.z := referenceData.reference_z +- .....;
ENDTRAP
```

Argumente

```
CapAPTrSetupPERS var_y, var_z, ReferenceData [\MaxIncrCorr]
     [\WarnMaxCorr] [\Filter] [\SampleTime] [\Logfile] [\LogSize]
     [\LatestCorr] [\AccCorr]
```

`var_y`

Datentyp: `num`

Analoges Eingangssignal, das als Prozessposition für die y-Richtung verwendet wird.

`var_z`

Datentyp: `signalai`

Analoges Eingangssignal, das als Prozessposition für die z-Richtung verwendet wird.

`ReferenceData`

Datentyp: `capaptrreferencedata`

Für die Korrekturreglerschleife verwendete Setup-Daten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.31 CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird.

Continuous Application Platform (CAP)
Fortsetzung

`MaxIncCorr`

Datentyp: `num`

Zulässige maximale inkrementelle Korrektur (in mm).

Wenn die inkrementelle TCP-Korrektur größer als `\MaxIncCorr` und `\WarnMaxCorr` ist, setzt der Roboter zwar seine Bahn fort, die übernommene inkrementelle Korrektur überschreitet jedoch nicht `\MaxIncCorr`. Wenn `\WarnMaxCorr` nicht angegeben ist, wird ein Verfolgungsfehler gemeldet und die Programmausführung angehalten.

`WarnMaxCorr`

Datentyp: `switch`

Wenn dieser Schalter vorhanden ist, wird die Programmausführung nicht unterbrochen, wenn der Grenzwert für die maximale Korrektur überschritten wird, angegeben in `\MaxIncCorr`. Es wird lediglich eine Warnung ausgegeben.

`Filter`

Datentyp: `num`

Größe des Referenz-Beispieldatenfilters. Ein Wert zwischen 1 und 15 ist erlaubt; der Standardwert ist 1.

`SampleTime`

Datentyp: `num`

Abtastrate in Millisekunden für die Korrekturschleife. Der Wert wird auf ein Vielfaches von 24 gerundet. Der zulässige Mindestwert ist 24 und der Standardwert ist 24.

`LogFile`

Datentyp: `string`

Der Name der tracklog-Protokolldatei. Die Protokolldatei wird im HOME -Verzeichnis des Systems abgelegt.

`LatestCorr`

Datentyp: `pos`

Größe der zuletzt hinzugefügten Korrektur (in mm).

`AccCorr`

Datentyp: `pos`

Größe der insgesamt akkumulierten Korrektur (in mm) hinzugefügt.

`LogSize`

Datentyp: `num`

Größe des tracklog-Ringpuffers, d. h. die Anzahl der Sensormessungen, die während der Verfolgung gepuffert werden können.

Standardwert: 1000.

Syntax

```
CapAPTrSetupPERS
  [var_y ':='] <expression (PERS) of num> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.31 CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird.

Continuous Application Platform (CAP)

Fortsetzung

```
[var_z ':='] <expression (PERS) of vnum> ','
[ReferenceData ':='] <expression (IN) of capaptrreferencedata>
     ','
[\MaxIncrCorr ':='] <expression (IN) of num> ','
[\WarnMaxCorr ':='] <expression (IN) of switch> ','
[\Filter ':='] <expression (IN) of num> ','
[\SampleTime ':='] <expression (IN) of num> ','
[\LogFile ':='] <expression (IN) of string> ','
[\LogSize ':='] <expression (IN) of num> ','
[\LatestCorr ':='] <expression (PERS) of pos> ','
[\AccCorr ':='] <expression (PERS) of pos> ';'
```

Weitere Informationen

Information über	Siehe
Instruktion `CapAPTrSetupAI`	*CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird. auf Seite 86*
Instruktion `CapAPTrSetupAO`	*CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird. auf Seite 89*
Datentyp `capaptrreferencedata`	*capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker auf Seite 1651*
Continuous Application Platform	*Application manual - Continuous Application Platform*
Sensor Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.32 CapC - Instruktion für kreisförmige CAP-Bewegung

Verwendung

`CapC` wird verwendet, um den Werkzeugmittelpunkt (TCP) entlang einer Kreisbahn zu einem gegebenen Ziel zu bewegen und zugleich einen kontinuierliches Prozess zu steuern. Weiterhin ist es möglich, bis zu acht Ereignisse an `CapC` anzuschließen. Die Ereignisse werden mit den folgenden Instruktionen definiert: `TriggRampAO` . `TriggIO` . `TriggEquip` . `TriggInt` . `TriggCheckIO` oder `TriggSpeed` .

Grundlegende Beispiele

Beispiel 1:

Kreisförmige Bewegungen mit `CapC`.

```
CapC cirp, p1, v100, cdata, weavestart, weave, fine, gun1;
```

Der TCP des Werkzeugs `gun1` wird kreisförmig zum Feinpunkt `p1` bewegt, und zwar mit der in`cdata` festgelegten Geschwindigkeit.

Beispiel 2:

Kreisförmige Bewegung bei Benutzerereignis und CAP-Ereignis.

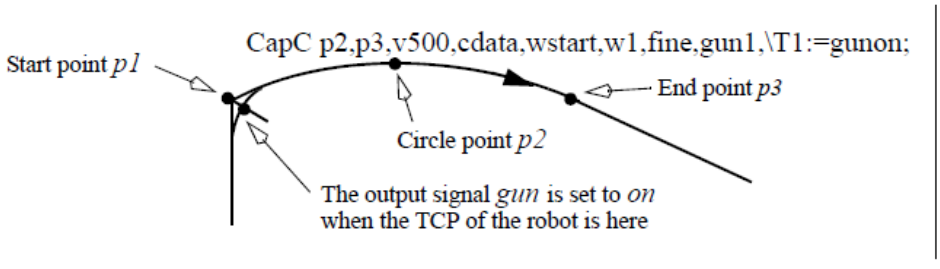
```
VAR intnum start_intno;
...
PROC main()
  VAR triggdata gunon;

  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
  TriggIO gunon, 0 \Start \DOp:=gun, on;

  MoveJ p1, v500, z50, gun1;
  CapC p2,p3,v500,cdata,wstart,w1,fine,gun1,\T1:=gunon;
ENDPROC

TRAP start_trap
  ! This routine will be executed when the event CAP_START is
       reported
ENDTRAP
```

Das digitale Ausgangssignal gun wird gesetzt, wenn der TCP die Mitte der Zonenbahn des Punkts `p1` erreicht. Die Trap-Routine start_trap wird ausgeführt, wenn der CAP-Prozess startet.



xx1200000174

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.32 CapC - Instruktion für kreisförmige CAP-Bewegung

Continuous Application Platform (CAP)
Fortsetzung

Argumente

```
CapC Cirpoint ToPoint [\Id] Speed Cdata [\MoveStartTimer] Weavestart
     Weave Zone [\Inpos] Tool [\WObj] [\Track] | [\Corr]
     [\PreProcessTracking] [\Time] [\T1] [\T2] [\T3] [\T4] [\T5]
     [\T6] [\T7] [\T8] [\TLoad]
```

`Cirpoint`

Datentyp: robtarget

Der Kreispunkt des Roboters. Beim Kreispunkt handelt es sich um einen Punkt zwischen dem Startpunkt und dem Zielpunkt. Damit eine optimale Genauigkeit erreicht wird, sollte dieser Punkt ungefähr auf halbem Weg zwischen dem Start- und Zielpunkt liegen. Wenn sich der Punkt zu nahe am Start- oder Endpunkt befindet, gibt der Roboter u. U. eine Warnung aus. Der Kreispunkt wird als eine benannte Position definiert oder direkt in der Instruktion gespeichert (wenn in der Instruktion mit * markiert).

`ToPoint`

Datentyp: robtarget

Der Zielpunkt des Roboters und der zusätzlichen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[\Id]`

(Sync-Identität)

Datentyp: identno

Synchronisationsidentität für eine RAPID- Bewegungsinstruktion in einem MultiMove-System im Synchronmodus.

`Speed`

Datentyp: speeddata

Die Geschwindigkeitsdaten, die für Bewegungen ohne aktiven CAP-Prozess gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugmittelpunkt, die zusätzlichen Achsen und die Werkzeugumorientierung. Ist ein CAP-Prozess aktiv (nicht blockiert), dann definiert das Cdata-Argument die TCP-Geschwindigkeit.

`Cdata`

(CAP-Prozessdaten)

Datentyp: capdata

CAP-Prozessdaten, siehe detaillierte Beschreibung unter *capdata - CAP-Daten auf Seite 1653*.

`[\Movestart_timer]`

(Zeit in s)

Datentyp: num

Obergrenze für den Zeitunterschied zwischen der Anweisung des Prozessstarts und dem tatsächlichen Beginn der TCP-Bewegung des Roboters in einem MultiMove-System im Synchronmodus.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Weavestart`

(Pendelstartdaten)

Datentyp: weavestartdata

Pendelstartdaten für den CAP-Prozess, siehe detaillierte Beschreibung unter *weavestartdata - Pendelstartdaten auf Seite 1850*.

`Weave`

(Pendeldaten)

Datentyp: capweavedata

Pendeldaten für den CAP-Prozess, siehe detaillierte Beschreibung unter *capweavedata - Pendeldaten für CAP auf Seite 1667*.

`Zone`

Datentyp: zonedata

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[\Inpos]`

(In Position)

Datentyp: stoppointdata

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: tooldata

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt (TCP) ist der Punkt, der zur angegebenen Zielposition bewegt wird.

`[\WObj]`

Datentyp: wobjdata

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden. In diesem Fall ist die Position dem Weltkoordinatensystem zugeordnet. Wenn allerdings ein stationärer TCP oder koordinierte zusätzliche Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[\Track]`

(Tracksensordaten)

Datentyp: captrackdata

Diese Datenstruktur enthält Daten für die eine Bahnkorrektur generierenden Sensoren gemeinsam mit `CapC`, siehe *captrackdata - CAP-Track-Daten auf Seite 1664*. Dieses Argument ist gemeinsam mit dem Argument `\Corr` nicht erlaubt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.32 CapC - Instruktion für kreisförmige CAP-Bewegung

Continuous Application Platform (CAP)
Fortsetzung

`[\Corr]`

(Korrekturgenerator verwenden)

Datentyp: switch

Dieses Argument weist `CapC` zum Lesen von Bahnkorrekturen aus einem Korrekturgenerator an, `CorrCon`. Dieses Argument ist nicht gemeinsam mit dem Argument `\Track` erlaubt.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[\PreProcessTracking]`

Datentyp: switch

Dieses Argument ist nur wirksam, wenn `first_instruction` auf `TRUE` gesetzt wird und das `\Track`-Argument vorhanden ist.

Dieses Argument aktiviert das *Pre Process Tracking* (Vorprozessverfolgung), was bedeutet, dass der Roboter die Verfolgung nur während dieser CapX-Instruktion durchführt, und zwar ohne Verarbeitung. Dadurch stehen gleich von Anfang der Bahn an Sensordaten für ein erfolgreiches Tracking mit Verarbeitung, z. B. Schweißen, zur Verfügung.

Weitere Informationen finden Sie unter *Operating manual - Tracking and searching with optical sensors*.

`[\Time]`

Datentyp: num

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der der Roboter und die zusätzlichen Achsen sich bewegen. Es wird dann gegen die entsprechenden Geschwindigkeitsdaten ausgetauscht.

`[\T1] [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8]`

(Trigg x)

Datentyp: triggdata

Variable, die sich auf Triggerbedingungen und Triggeraktivitäten bezieht, früher im Programm mit den Instruktionen `TriggRampAO`, `TriggIO`, `TriggEquip` oder `TriggInt` definiert.

`[\TLoad]`

Datentyp: loaddata

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das Argument `\TLoad` auf `load0` gesetzt ist, dann wird das Argument nicht berücksichtigt, stattdessen wird das `loaddata` im aktuellen `tooldata` verwendet. Für eine vollständige Beschreibung des Arguments `TLoad`, siehe `MoveL`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Informationen über die lineare Bewegung finden Sie unter *MoveL - Lineare Bewegung des Roboters auf Seite 448*.

Informationen über die lineare Bewegung mit Trigger-Ereignissen finden Sie unter *TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*.

Fehlerbehandlung

Es gibt verschiedene Arten von Fehlern, die von der Fehlerbehandlungsroutine für die `CapC` / `CapL` Instruktion behandelt werden können:

- Überwachungsfehler
- Sensorspezifische Fehler
- Für ein MultiMove-System spezifische Fehler
- Fehler übernommen von der `TriggX` Funktionalität
- andere CAP-Fehler

Wenn eins der Signale, die überwacht werden sollen, einen falschen Wert hat oder sich der Wert während der Überwachung ändert, wird die Systemvariable `ERRNO` gesetzt.

Wenn keine Werte aus dem Verfolgungssensor gelesen werden können, wird die Systemvariable `ERRNO` gesetzt.

Für das MultiMove-System im Synchronmodus muss die Fehlerbehandlung zwei weitere Fehler beachten. Einer davon meldet, dass eine andere Anwendung einen behebbaren Fehler festgestellt hat. Das ermöglicht die Fehlerbehandlung eines behebbaren Fehlers von synchronisierten RAPID-Tasks. Der andere Fehler, `CAP_MOV_WATCHDOG` meldet, dass die Zeit zwischen der Reihenfolge des Prozessstarts und dem tatsächlichen Beginn der Bewegung des TCP des Roboters in einem MultiMove-System im Synchronmodus abläuft. Die verwendete Zeit wird im optionalen Parameter `Movestart_timer` in der Instruktion `CapC` angegeben.

Bei Feststellung einer Störung wird die Programmabarbeitung angehalten. Ist jedoch eine Fehlerbehandlungsroutine programmiert, können die nachstehend angegebenen Fehler ohne Produktionsunterbrechung behoben werden. Es empfiehlt sich jedoch, dem Endbenutzer einige Fehler (die Fehler mit `CAP_XX`) nicht zu präsentieren. Ordnen Sie diese Fehler den anwendungsspezifischen Fehlern zu. Für die Überwachungsfehler kann mit der Instruktion `CapGetFailSigs` abgerufen werden, welches spezifische Signal fehlgeschlagen ist.

Überwachungsfehler

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`CAP_PRE_ERR`	Dieser Fehler tritt bei einem Fehler in der `PRE` Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in `pre_cond` Timeout).
`CAP_PRESTART_ERR`	Dieser Fehler tritt bei einem Fehler während der Überwachung der Phase `PRE` auf.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.32 CapC - Instruktion für kreisförmige CAP-Bewegung

Continuous Application Platform (CAP)

Fortsetzung

CAP_END_PRE_ERR	Dieses Ereignis tritt bei einem Fehler in der END_PRE Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in start_cond Timeout).
CAP_START_ERR	Dieses Ereignis tritt bei einem Fehler in der START Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in start_cond Timeout).
CAP_MAIN_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Hauptphase auf.
CAP_ENDMAIN_ERR	Dieser Fehler tritt bei einem Fehler in der END_MAIN Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_START_POST1_ERR	Dieses Ereignis tritt bei einem Fehler in der START_POST1 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_POST1_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Phase POST1 auf.
CAP_POST1END_ERR	Dieser Fehler tritt bei einem Fehler in der END_POST1 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_START_POST2_ERR	Dieses Ereignis tritt bei einem Fehler in der START_POST1 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_POST2_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Phase POST2 auf.
CAP_POST2END_ERR	Dieser Fehler tritt bei einem Fehler in der END_POST2 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout). Wenn die Überwachung für zwei verschiedene Signale in der gleichen Phase durchgeführt wird und beide ausfallen, ist es das erste, das mit SetupSuperv eingerichtet ist, das den Fehler erzeugt.

Sensorbezogene Fehler

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

CAP_TRACK_ERR	Der Verfolgungsfehler tritt beim Lesen der Daten des Sensors auf und wenn über einen bestimmten Zeitraum keine gültigen Daten empfangen werden. Ein Grund dafür könnte sein, dass der Sensor die Naht nicht anzeigen kann.
CAP_TRACKSTA_ERR	Der Verfolgungsfehler tritt auf, wenn der Laser-Verfolgungssensor keine gültigen Daten gelesen hat.
CAP_TRACKCOR_ERR	Der Laserkorrekturfehler tritt auf, wenn bei der Berechnung der Verschiebung ein Fehler auftritt.
CAP_TRACKCOM_ERR	Die Kommunikation zwischen der Robotersteuerung und der Sensorausrüstung wurde abgebrochen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

CAP_TRACKPFR_ERR	Es ist nicht möglich, die Verfolgung fortzusetzen, weil es während der Verfolgung zu einem Stromausfall gekommen ist.
CAP_SEN_NO_MEAS	Die Steuerung hat keine gültige Messung vom Sensor erhalten.
CAP_SEN_NOREADY	Der Sensor ist noch nicht bereit.
CAP_SEN_GENERRO	Es ist ein allgemeiner Sensorfehler aufgetreten.
CAP_SEN_BUSY	Der Sensor ist ausgelastet und kann die Anfrage nicht beantworten.
CAP_SEN_UNKNOWN	Der Befehl, der an den Sensor geschickt wurde, ist dem Sensor nicht bekannt.
CAP_SEN_ILLEGAL	Die an den Sensor geschickte Variable oder Blocknummer ist illegal.
CAP_SEN_EXALARM	Im Sensor wurde ein externer Alarm ausgelöst.
CAP_SEN_CAALARM	Im Sensor wurde ein Kameraalarm ausgelöst.
CAP_SEN_TEMP	Die Sensortemperatur liegt außerhalb des zulässigen Bereichs.
CAP_SEN_VALUE	Der Wert, der an den Sensor gesendet wird, liegt außerhalb des Bereichs.
CAP_SEN_CAMCHECK	Die Kameraprüfung ist fehlgeschlagen.
CAP_SEN_TIMEOUT	Der Sensor hat innerhalb des Timeouts nicht reagiert.

Mögliche Fehler im MultiMove-System

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_PATH_STOP	Wenn die synchronisierte Bewegung verwendet wird, wird dieser Fehler gemeldet, wenn eine Anwendung zur Steuerung einer Einheit einen behebbaren Fehler erkennt und anderen Anwendungen meldet, dass ein Fehler aufgetreten ist. Wird der Fehlercode von einer CapC Instruktion empfangen, ist der Fehler eine Reaktion auf einen anderen Fehler. Alle Tasks unter Verwendung von Bewegungsinstruktionen im Synchronmodus in einem MultiMove-System müssen diesen ERRNO Wert in der Fehlerbehandlung definiert haben.

Fehler übernommen von TriggX

Die Instruktion CapC basiert auf der Instruktion TriggC. Als Folge können Sie die Fehlermeldungen ERR_AO_LIM und ERR_DIPLAG_LIM wie in TriggC abrufen und handhaben.

Die Systemvariable ERRNO wird gesetzt auf:

ERR_AO_LIM	Wenn das programmierte ScaleValue/SetValue-Argument für das angegebene analoge Ausgangssignal AOp/AOutput in einigen der verbundenen TriggSpeed/TriggRampAO-Instruktionen zusammen mit dem programmierten Speed-Argument in dieser Instruktion außerhalb der Grenzen für das analoge Signal liegt, wird die Systemvariable ERRNO auf ERR_AO_LIM gesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ERR_DIPLAG_LIM	Wenn das programmierte DipLag-Argument in einer verbundenen TriggSpeed-Instruktion im Verhältnis zu den Systemparametern *Event Preset Time* zu groß ist, wird die Systemvariable ERRNO auf ERR_DIPLAG_LIM gesetzt.

Andere CAP-Fehler

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

CAP_ATPROC_START	Dieser behebbarer Fehler wird am Ende des ersten CapC/L-Instruktion einer Sequenz erzeugt, wenn das optionale Argument \PreProcessTracking verwendet wird. Er kann dann in der Fehlerbehandlungsroutine behoben werden, um den Prozess zu starten. Weitere Informationen finden Sie unter *Operating manual - Tracking and searching with optical sensors*.
CAP_NOPROC_END	Dieser Fehler tritt auf, wenn die Instruktion CapNoProcess verwendet wird, um eine bestimmte Entfernung ohne Anwendungsprozess zurückzulegen und das Ende dieser Entfernung erreicht ist. Das ist kein echter Fehler, es werden jedoch die Mechanismen der Fehlerkorrektur verwendet.
CAP_MOV_WATCHDOG	Dieser Fehler tritt auf, wenn der Schalter \Movestart_timer angegeben ist und die Zeit zwischen dem Prozessstart (MAIN_STARTED) und dem Start der Roboterbewegung die für den Schalter angegebene Zeit überschreitet.

CAP-Prozess

Bei der kontinuierlichen Ausführung sowohl des Auto-Modus als auch des manuellen Modus wird der CAP-Prozess ausgeführt, es sei denn, er wird blockiert. Das bedeutet, dass alle Daten der CAP-Prozesssteuerung (das heißt, Cdata . Weavestart . Weave und Movestart_timer) verwendet werden. In diesem Modus werden alle CAP-Triggeraktivitäten ausgeführt, siehe *ICap - verbindet CAP-Ereignisse mit Trap-Routinen auf Seite 246*.

In allen anderen Ausführungsmodi wird der CAP-Prozess nicht ausgeführt und die CapC Instruktion verhält sich wie eine MoveC Instruktion.

Triggerbedingungen [\T1] bis [\T8]

Wenn die Triggerbedingungen erfüllt sind, während sich der Roboter dem Endpunkt nähert, werden die definierten Triggeraktivitäten ausgeführt. Die Triggerbedingungen sind entweder einen bestimmten Abstand vor dem Endpunkt der Instruktion oder an einem eingestellten Abstand nach dem Startpunkt der Instruktion oder zu einem bestimmten Zeitpunkt (auf eine kurze Zeit begrenzt) vor dem Endpunkt der Instruktion erfüllt.

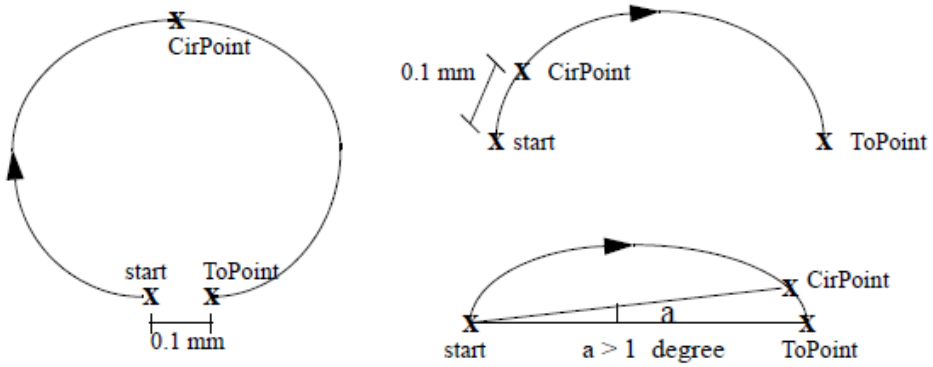
Bei der schrittweisen Vorwärtsabarbeitung werden die E/A-Aktivitäten ausgeführt, aber die Interrupt-Routinen laufen nicht ab. Bei der schrittweisen Rückwärtsabarbeitung werden überhaupt keine Triggeraktivitäten ausgeführt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Es gibt einige Einschränkungen in der Platzierung von *CirPoint* und *ToPoint*, wie in nachstehender Abbildung gezeigt.



xx1200000175

- Mindestabstand zwischen Start und ToPoint beträgt 0,1 mm.
- Mindestabstand zwischen Start und CirPoint beträgt 0,1 mm.
- Minimaler Winkel zwischen CirPoint und ToPoint vom Startpunkt beträgt 1 Grad.

Die Genauigkeit an den Grenzen kann gering sein; wenn z. B. Startpunkt und ToPoint auf dem Kreisbogen nahe beieinander liegen, ist der Fehler durch die Neigung des Kreises viel größer als die Genauigkeit, mit der die Punkte programmiert wurden.

Eine Änderung des Abarbeitungsmodus von vorwärts in rückwärts oder umgekehrt, während der Roboter auf einer kreisförmigen Bahn stoppt, ist nicht zulässig und verursacht eine Fehlermeldung.

Die Instruktion `CapC` (oder jede andere Instruktion mit kreisförmiger Bewegung) darf nie von Beginn an neu gestartet werden, wenn sich der TCP zwischen dem Kreispunkt und dem Endpunkt befindet. In diesem Fall arbeitet der Roboter die programmierte Bahn nicht weiter ab (die Positionierung an der kreisförmigen Bahn erfolgt nicht in der programmierten Richtung).

Stellen Sie sicher, dass der Roboter während der Programmabarbeitung den Kreispunkt erreicht und das Kreissegment ggf. unterteilen kann.

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion `CapC` kürzer als üblich ist, sind eventuell einige oder alle Triggerbedingungen sofort und in derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine „unvollständige Bewegung“ nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

`CapC` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.32 CapC - Instruktion für kreisförmige CAP-Bewegung

Continuous Application Platform (CAP)
Fortsetzung

Syntax

```
CapC
  [CirPoint ':='] < expression (IN) of robtarget >
  [ToPoint ':='] < expression (IN) of robtarget >
  ['\' Id ':=' < expression (IN) of identno > ] ','
  [Speed ':='] < expression (IN) of speeddata >
  [Cdata ':='] < persistent (PERS) of capdata >
  ['\' Movestart_timer ':=' < expression (IN) of num > ] ','
  [Weavestart ':='] <persistent (PERS) of weavestartdata >
  [Weave ':='] < persistent (PERS) of capweavedata >
  [Zone ':='] < expression (IN) of zonedata >
  ['\' Inpos ':=' < expression (IN) of stoppointdata >] ','
  [Tool ':='] < persistent (PERS) of tooldata >
  ['\' WObj ':=' < persistent (PERS) of wobjdata > ]
  ['\' Track ':=' < persistent (PERS) of captrackdata > ]
  |[ '\' Corr]
  |[ '\' PreProcessTracking]
  ['\' Time ':=' < expression (IN) of num > ]
  ['\' T1 ':=' < variable (VAR) of triggdata > ]
  ['\' T2 ':=' < variable (VAR) of triggdata > ]
  ['\' T3 ':=' < variable (VAR) of triggdata > ]
  ['\' T4 ':=' < variable (VAR) of triggdata > ]
  ['\' T5 ':=' < variable (VAR) of triggdata > ]
  ['\' T6 ':=' < variable (VAR) of triggdata > ]
  ['\' T7 ':=' < variable (VAR) of triggdata > ]
  ['\' T8 ':=' < variable (VAR) of triggdata > ]
  ['\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Kreisbewegung	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Kreisförmige Bewegung mit Triggern	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Definition der CAP-Daten	*capdata - CAP-Daten auf Seite 1653*
Definition von Pendelstartdaten	*weavestartdata - Pendelstartdaten auf Seite 1850*
Definition von Pendeldaten	*capweavedata - Pendeldaten für CAP auf Seite 1667*
Definition von Track-Daten	*captrackdata - CAP-Track-Daten auf Seite 1664*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*
Verwendung von optischen Sensoren für Nachverfolgung oder Suche.	*Operating manual - Tracking and searching with optical sensors*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.33 CapCondSetDO - Setzt ein digitales Ausgangssignal bei TCP-Stopp

Verwendung

`CapCondSetDO` wird verwendet, um ein digitales Ausgangssignal und dessen Wert zu definieren, die eingestellt werden, wenn der TCP des Roboters, der CAP ausführt, während einer CAP-Instruktion die Bewegung stoppt (`CapL` oder `CapC`), bevor die CAP-Sequenz beendet ist.

Eine bestehende Definition dieser Signale wird mit der CAP-Instruktion `InitSuperv` gelöscht.

Grundlegendes Beispiel

```
CapCondSetDO do15, 1;
```

Das Ausgangssignal do15 wird auf 1 gesetzt, wenn der TCP stoppt.

```
CapCondSetDO weld, off;
```

Das Signal Schweißen wird auf `off` gesetzt, wenn der TCP stoppt.

Argumente

```
CapCondSetDO Signal Value
```

Signal

Datentyp: `signaldo`

Der Name des zu ändernden Signals.

Value

Datentyp: `dionum`

Der gewünschte Signalwert (0 oder 1).

Angegebener Value	Setzt digitalen Ausgang auf
0	0
Beliebiger Wert außer 0	1

Einschränkungen

Der finale Wert hängt von der Konfiguration des Signals ab. Wenn das Signal in den Systemparametern invertiert ist, hat der physische Kanal den entgegengesetzten Wert.

Es können maximal 10 Signale pro RAPID-Task festgelegt werden.

Syntax

```
CapCondSetDO
  [Signal ':='] < variable (VAR) of signaldo > ','
  [Value ':='] < expression (IN) of dionum > ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.33 CapCondSetDO - Setzt ein digitales Ausgangssignal bei TCP-Stopp

Continuous Application Platform (CAP)

Fortsetzung

Information über	Siehe
Instruktion `InitSuperv`	*InitSuperv - Zurücksetzen der gesamten CAP-Überwachung auf Seite 284*
Instruktion `SetupSuperv`	*SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP auf Seite 725*
Instruktion `RemoveSuperv`	*RemoveSuperv Bedingung für ein Signal entfernen auf Seite 607*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.34 CapEquiDist Ereignis mit gleichem Abstand erstellen

Verwendung

`CapEquiDist` wird verwendet, um CAP darüber in Kenntnis zu setzen, dass auf dem CAP-Pfad ein RAPID-Ereignis mit gleichem Abstand (EQUIDIST) erzeugt werden soll. Das erste Ereignis wird am Startpunkt der ersten CAP-Instruktion in einer Sequenz von CAP-Instruktionen generiert. Von RAPID ist es möglich, dieses Ereignis mithilfe von`ICap` zu abonnieren.

Grundlegendes Beispiel

```
VAR intnum intno_equi;

PROC main()
  ......
  IDelete intno_equi;
  Connect intno_equi equi_trp;
  ICap intno_equi, EQUIDIST
  ......
  CapEquiDist\Distance:=5.0;
  MoveL p60, v1000, fine, tWeldGun;
  CapL p_fig3_l_1, v500, cd, wsd, cwd, z10, tWeldGun;
  CapL p_fig3_l_2, v500, cd, wsd, cwd, fine, tWeldGun;
  ......
  CapEquiDist\Reset;
  MoveL p70, v1000, fine, tWeldGun;
  CapL p_fig3_l_3, v500, cd, wsd, cwd, fine, tWeldGun;
  ......

  ERROR
    Retry;
ENDPROC

TRAP equi_trp
  ! do whatever you want, but it must not take too long time
ENDTRAP
```

In diesem Beispiel wird das Ereignis `EQUIDIST` auf dem ersten CAP-Pfad erzeugt. Es wird alle 5 mm der Bahn über mehrere CAP-Instruktionen mit Zonen gesendet.

Argumente

```
CapEquiDist [\Distance] [\Reset]
```

[\Distance]

Abstand in mm

Datentyp: `num`

Die Daten, die dieses optionale Argument bereitstellt, definieren den Abstand zwischen zwei aufeinanderfolgenden Ereignissen mit gleichem Abstand.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.34 CapEquiDist Ereignis mit gleichem Abstand erstellen

Continuous Application Platform (CAP)
Fortsetzung

[\Reset]

Ereigniserzeugung zurücksetzen

Datentyp: `switch`

Ist dieser Schalter vorhanden, wird die Ereigniserzeugung zurückgesetzt, das heißt, das Ereignis mit gleichem Abstand wird auf einer`CapL`/`CapC`-Bahn nicht mehr generiert. Dieser Schalter hat Vorrang vor dem Schalter `\Distance`.

Einschränkungen

Wenn die CAP-Bahn im Vergleich zum Abstand des Ereignisses lang ist, können die Ereignis-Ressourcen des Systems aufgebraucht sein und es erscheint die Fehlermeldung **50368 Too Short distance between equidistant events**(50368 Zu geringer Abstand zwischen Ereignissen mit gleichem Abstand).

Syntax

```
CapEquiDist
  ['\' Distance ':=' < expression (IN) of num >]
  ['\' Reset] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.35 CapL - Instruktion für lineare CAP-Bewegung

Verwendung

`CapL` wird verwendet, um den Werkzeugmittelpunkt (TCP) linear zu einem gegebenen Ziel zu bewegen und zugleich einen kontinuierlichen Prozess zu steuern. Weiterhin ist es möglich, bis zu acht Ereignisse an `CapL` anzuschließen. Die Ereignisse werden mit den folgenden Instruktionen definiert: `TriggRampAO`, `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO` oder `TriggSpeed`.

Grundlegende Beispiele

Beispiel 1

Lineare Bewegungen mit `CapL`.

```
CapL p1, v100, cdata, weavestart, weave, z50, gun1;
```

Der TCP des Werkzeugs gun1 wird linear in Richtung der Position p1 bewegt, und zwar mit der in `cdata` festgelegten Geschwindigkeit und den Zonendaten z50.

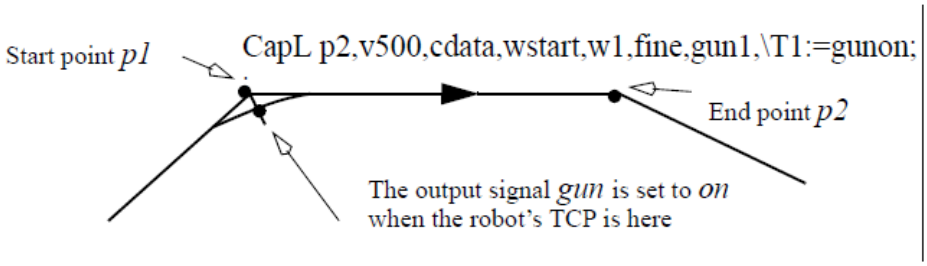
Beispiel 2:

Kreisförmige Bewegung bei Benutzerereignis und CAP-Ereignis.

```
VAR intnum start_intno;
...
PROC main()
  VAR triggdata gunon;
  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
  TriggIO gunon, 0 \Start \DOp:=gun, on;
  MoveJ p1, v500, z50, gun1;
  CapL p2, v500, cdata, wstart, w1, fine, gun1 \T1:=gunon;
ENDPROC

TRAP start_trap
  !This routine is executed when event CAP_START arrives
ENDTRAP
```

Das digitale Ausgangssignal gun wird gesetzt, wenn der TCP die Mitte der Zonenbahn des Punkts p1 erreicht. Die Trap-Routine `start_trap` wird ausgeführt, wenn der CAP-Prozess startet.



xx1200000173

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.35 CapL - Instruktion für lineare CAP-Bewegung

Continuous Application Platform (CAP)
Fortsetzung

Argumente

```
CapLToPoint [\Id] Speed Cdata [\MoveStartTimer] Weavestart Weave
    Zone [\Inpos] Tool [\WObj] [\Track] | [\Corr]
    [\PreProcessTracking] [\Time] [\T1] [\T2] [\T3] [\T4] [\T5]
    [\T6] [\T7] [\T8] [\TLoad]
```

`ToPoint`

Datentyp: robtarget

Der Zielpunkt des Roboters und der zusätzlichen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \Id ]`

(Sync-Identität)

Datentyp: identno

Synchronisationsidentität für eine RAPID- Bewegungsinstruktion in einem MultiMove-System im Synchronmodus.

`Speed`

Datentyp: speeddata

Die Geschwindigkeitsdaten, die für Bewegungen ohne aktiven CAP-Prozess gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugmittelpunkt, die zusätzlichen Achsen und die Werkzeugumorientierung. Ist ein CAP-Prozess aktiv (nicht blockiert), dann definiert das Cdata-Argument die TCP-Geschwindigkeit.

`Cdata`

(CAP-Prozessdaten)

Datentyp: capdata

CAP-Prozessdaten, siehe detaillierte Beschreibung unter *capdata - CAP-Daten auf Seite 1653*.

`[ \Movestart_timer ]`

(Zeit in s)

Datentyp: num

Obergrenze für den Zeitunterschied zwischen der Anweisung des Prozessstarts und dem tatsächlichen Beginn der TCP-Bewegung des Roboters in einem MultiMove-System im Synchronmodus.

`Weavestart`

(Pendelstartdaten)

Datentyp: weavestartdata

Pendelstartdaten für den CAP-Prozess, siehe detaillierte Beschreibung unter *weavestartdata - Pendelstartdaten auf Seite 1850*.

`Weave`

(Pendeldaten)

Datentyp: capweavedata

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Pendeldaten für den CAP-Prozess, siehe detaillierte Beschreibung unter *capweavedata - Pendeldaten für CAP auf Seite 1667*.

`Zone`

Datentyp: zonedata

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[\Inpos ]`

(In Position)

Datentyp: stoppointdata

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: tooldata

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt (TCP) ist der Punkt, der zur angegebenen Zielposition bewegt wird.

`[ \WObj ]`

Datentyp: wobjdata

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden. In diesem Fall ist die Position dem Weltkoordinatensystem zugeordnet. Wenn allerdings ein stationärer TCP oder koordinierte zusätzliche Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \Track ]`

(Tracksensordaten)

Datentyp: captrackdata

Diese Datenstruktur enthält Daten für die eine Bahnkorrektur generierenden Sensoren gemeinsam mit `CapL`, siehe *captrackdata - CAP-Track-Daten auf Seite 1664*. Dieses Argument ist gemeinsam mit dem Argument `\Corr` nicht erlaubt.

`[ \Corr ]`

(Korrekturgenerator verwenden)

Datentyp: switch

Dieses Argument weist `CapL` zum Lesen von Bahnkorrekturen aus einem Korrekturgenerator an, `CorrCon`. Dieses Argument ist nicht gemeinsam mit dem Argument `\Track` erlaubt.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.35 CapL - Instruktion für lineare CAP-Bewegung
Continuous Application Platform (CAP)
Fortsetzung

`[\PreProcessTracking]`

Datentyp: switch

Dieses Argument ist nur wirksam, wenn `first_instruction` auf `TRUE` gesetzt wird und das `\Track`-Argument vorhanden ist.

Dieses Argument aktiviert das *Pre Process Tracking* (Vorprozessverfolgung), was bedeutet, dass der Roboter die Verfolgung nur während dieser CapX-Instruktion durchführt, und zwar ohne Verarbeitung. Dadurch stehen gleich von Anfang der Bahn an Sensordaten für ein erfolgreiches Tracking mit Verarbeitung, z. B. Schweißen, zur Verfügung.

Weitere Informationen finden Sie unter *Operating manual - Tracking and searching with optical sensors*.

`[ \Time ]`

Datentyp: num

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der der Roboter und die zusätzlichen Achsen sich bewegen. Es wird dann gegen die entsprechenden Geschwindigkeitsdaten ausgetauscht.

`[ \T1 ]` bis `[ \T8 ]`

(Trigg x)

Datentyp: triggdata

Variable, die sich auf Triggerbedingungen und Triggeraktivitäten bezieht, früher im Programm mit den Instruktionen `TriggRampAO`, `TriggIO`, `TriggEquip` oder `TriggInt` definiert.

`[\TLoad]`

Datentyp: loaddata

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das Argument `\TLoad` auf `load0` gesetzt ist, dann wird das Argument nicht berücksichtigt, stattdessen wird das `loaddata` im aktuellen `tooldata` verwendet. Für eine vollständige Beschreibung des Arguments `TLoad`, siehe `MoveL`.

Programmabarbeitung

Informationen über die lineare Bewegung finden Sie unter *MoveL - Lineare Bewegung des Roboters auf Seite 448*.

Informationen über die lineare Bewegung mit Trigger-Ereignissen finden Sie unter *TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*.

Fehlerbehandlung

Es gibt verschiedene Arten von Fehlern, die von der Fehlerbehandlungsroutine für die `CapC` / `CapL` Instruktion behandelt werden können:

- Überwachungsfehler
- Sensorspezifische Fehler

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Für ein MultiMove-System spezifische Fehler
- Fehler übernommen von der TriggX Funktionalität
- andere CAP-Fehler

Wenn eins der Signale, die überwacht werden sollen, einen falschen Wert hat oder sich der Wert während der Überwachung ändert, wird die Systemvariable ERRNO gesetzt.

Wenn keine Werte aus dem Verfolgungssensor gelesen werden können, wird die Systemvariable ERRNO gesetzt.

Für das MultiMove-System im Synchronmodus muss die Fehlerbehandlung zwei weitere Fehler beachten. Einer davon meldet, dass eine andere Anwendung einen behebbaren Fehler festgestellt hat. Das ermöglicht die Fehlerbehandlung eines behebbaren Fehlers von synchronisierten RAPID-Tasks. Der andere Fehler, CAP_MOV_WATCHDOG meldet, dass die Zeit zwischen der Reihenfolge des Prozessstarts und dem tatsächlichen Beginn der Bewegung des TCP des Roboters in einem MultiMove-System im Synchronmodus abläuft. Die verwendete Zeit wird im optionalen Parameter Movestart_timer in der Instruktion CapL angegeben.

Bei Feststellung einer Störung wird die Programmabarbeitung angehalten. Ist jedoch eine Fehlerbehandlungsroutine programmiert, können die nachstehend angegebenen Fehler ohne Produktionsunterbrechung behoben werden. Es empfiehlt sich jedoch, dem Endbenutzer einige Fehler (die Fehler mit CAP_XX) nicht zu präsentieren. Ordnen Sie diese Fehler den anwendungsspezifischen Fehlern zu. Für die Überwachungsfehler kann mit der Instruktion CapGetFailSigs abgerufen werden, welches spezifische Signal fehlgeschlagen ist.

Überwachungsfehler

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

CAP_PRE_ERR	Dieser Fehler tritt bei einem Fehler in der PRE Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in pre_cond Timeout).
CAP_PRESTART_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Phase PRE auf.
CAP_END_PRE_ERR	Dieses Ereignis tritt bei einem Fehler in der END_PRE Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in start_cond Timeout).
CAP_START_ERR	Dieses Ereignis tritt bei einem Fehler in der START Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in start_cond Timeout).
CAP_MAIN_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Hauptphase auf.
CAP_ENDMAIN_ERR	Dieser Fehler tritt bei einem Fehler in der END_MAIN Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.35 CapL - Instruktion für lineare CAP-Bewegung

Continuous Application Platform (CAP)

Fortsetzung

CAP_START_POST1_ERR	Dieses Ereignis tritt bei einem Fehler in der START_POST1 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_POST1_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Phase POST1 auf.
CAP_POST1END_ERR	Dieser Fehler tritt bei einem Fehler in der END_POST1 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_START_POST2_ERR	Dieses Ereignis tritt bei einem Fehler in der START_POST1 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout).
CAP_POST2_ERR	Dieser Fehler tritt bei einem Fehler während der Überwachung der Phase POST2 auf.
CAP_POST2END_ERR	Dieser Fehler tritt bei einem Fehler in der END_POST2 Überwachungsliste auf, das heißt, wenn die Bedingungen in der Liste nicht innerhalb des festgelegten Zeitrahmens erfüllt werden (angegeben in end_main_cond Timeout). Wenn die Überwachung für zwei verschiedene Signale in der gleichen Phase durchgeführt wird und beide ausfallen, ist es das erste, das mit SetupSuperv eingerichtet ist, das den Fehler erzeugt.

Sensorbezogene Fehler

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

CAP_TRACK_ERR	Der Verfolgungsfehler tritt beim Lesen der Daten des Sensors auf und wenn über einen bestimmten Zeitraum keine gültigen Daten empfangen werden. Ein Grund dafür könnte sein, dass der Sensor die Naht nicht anzeigen kann.
CAP_TRACKSTA_ERR	Der Verfolgungsfehler tritt auf, wenn der Laser-Verfolgungssensor keine gültigen Daten gelesen hat.
CAP_TRACKCOR_ERR	Der Laserkorrekturfehler tritt auf, wenn bei der Berechnung der Verschiebung ein Fehler auftritt.
CAP_TRACKCOM_ERR	Die Kommunikation zwischen der Robotersteuerung und der Sensorausrüstung wurde abgebrochen.
CAP_TRACKPFR_ERR	Es ist nicht möglich, die Verfolgung fortzusetzen, weil es während der Verfolgung zu einem Stromausfall gekommen ist.
CAP_SEN_NO_MEAS	Die Steuerung hat keine gültige Messung vom Sensor erhalten.
CAP_SEN_NOREADY	Der Sensor ist noch nicht bereit.
CAP_SEN_GENERRO	Es ist ein allgemeiner Sensorfehler aufgetreten.
CAP_SEN_BUSY	Der Sensor ist ausgelastet und kann die Anfrage nicht beantworten.
CAP_SEN_UNKNOWN	Der Befehl, der an den Sensor geschickt wurde, ist dem Sensor nicht bekannt.
CAP_SEN_ILLEGAL	Die an den Sensor geschickte Variable oder Blocknummer ist illegal.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

CAP_SEN_EXALARM	Im Sensor wurde ein externer Alarm ausgelöst.
CAP_SEN_CAALARM	Im Sensor wurde ein Kameraalarm ausgelöst.
CAP_SEN_TEMP	Die Sensortemperatur liegt außerhalb des zulässigen Bereichs.
CAP_SEN_VALUE	Der Wert, der an den Sensor gesendet wird, liegt außerhalb des Bereichs.
CAP_SEN_CAMCHECK	Die Kameraprüfung ist fehlgeschlagen.
CAP_SEN_TIMEOUT	Der Sensor hat innerhalb des Timeouts nicht reagiert.

Mögliche Fehler im MultiMove-System

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_PATH_STOP	Wenn die synchronisierte Bewegung verwendet wird, wird dieser Fehler gemeldet, wenn eine Anwendung zur Steuerung einer Einheit einen behebbaren Fehler erkennt und anderen Anwendungen meldet, dass ein Fehler aufgetreten ist. Wird der Fehlercode von einer CapL Instruktion empfangen, ist der Fehler eine Reaktion auf einen anderen Fehler. Alle Tasks unter Verwendung von Bewegungsinstruktionen im Synchronmodus in einem MultiMove-System müssen diesen ERRNO Wert in der Fehlerbehandlung definiert haben.

Fehler übernommen von TriggX

Die Instruktion CapL basiert auf der Instruktion TriggL. Als Folge können Sie die Fehlermeldungen ERR_AO_LIM und ERR_DIPLAG_LIM wie in TriggL abrufen und handhaben.

Die Systemvariable ERRNO wird gesetzt auf:

ERR_AO_LIM	Wenn das programmierte ScaleValue/SetValue-Argument für das angegebene analoge Ausgangssignal AOp/AOutput in einigen der verbundenen TriggSpeed/TriggRampAO-Instruktionen zusammen mit dem programmierten Speed-Argument in dieser Instruktion außerhalb der Grenzen für das analoge Signal liegt, wird die Systemvariable ERRNO auf ERR_AO_LIM gesetzt.
ERR_DIPLAG_LIM	Wenn das programmierte DipLag-Argument in einer verbundenen TriggSpeed-Instruktion im Verhältnis zu den Systemparametern *Event Preset Time* zu groß ist, wird die Systemvariable ERRNO auf ERR_DIPLAG_LIM gesetzt.

Andere CAP-Fehler

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

CAP_ATPROC_START	Dieser behebbarer Fehler wird am Ende des ersten CapC/L-Instruktion einer Sequenz erzeugt, wenn das optionale Argument \PreProcessTracking verwendet wird. Er kann dann in der Fehlerbehandlungsroutine behoben werden, um den Prozess zu starten. Weitere Informationen finden Sie unter *Operating manual - Tracking and searching with optical sensors*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.35 CapL - Instruktion für lineare CAP-Bewegung
Continuous Application Platform (CAP)
Fortsetzung

CAP_NOPROC_END	Dieser Fehler tritt auf, wenn die Instruktion CapNoProcess verwendet wird, um eine bestimmte Entfernung ohne Anwendungsprozess zurückzulegen und das Ende dieser Entfernung erreicht ist. Das ist kein echter Fehler, es werden jedoch die Mechanismen der Fehlerkorrektur verwendet.
CAP_MOV_WATCHDOG	Dieser Fehler tritt auf, wenn der Schalter \Movestart_timer angegeben ist und die Zeit zwischen dem Prozessstart (MAIN_STARTED) und dem Start der Roboterbewegung die für den Schalter angegebene Zeit überschreitet.

CAP-Prozess

Bei der kontinuierlichen Ausführung sowohl des Auto-Modus als auch des manuellen Modus wird der CAP-Prozess ausgeführt, es sei denn, er wird blockiert. Das bedeutet, dass alle Daten der CAP-Prozesssteuerung (das heißt, Cdata . Weavestart . Weave und Movestart_timer) verwendet werden. In diesem Modus werden alle CAP-Triggeraktivitäten ausgeführt, siehe *ICap - verbindet CAP-Ereignisse mit Trap-Routinen auf Seite 246*.

In allen anderen Ausführungsmodi wird der CAP-Prozess nicht ausgeführt und die CapL Instruktion verhält sich wie eine MoveL Instruktion.

Triggerbedingungen [\T1] bis [\T8]

Wenn die Triggerbedingungen erfüllt sind, während sich der Roboter dem Endpunkt nähert, werden die definierten Triggeraktivitäten ausgeführt. Die Triggerbedingungen sind entweder einen bestimmten Abstand vor dem Endpunkt der Instruktion oder an einem eingestellten Abstand nach dem Startpunkt der Instruktion oder zu einem bestimmten Zeitpunkt (auf eine kurze Zeit begrenzt) vor dem Endpunkt der Instruktion erfüllt.

Bei der schrittweisen Vorwärtsabarbeitung werden die E/A-Aktivitäten ausgeführt, aber die Interrupt-Routinen laufen nicht ab. Bei der schrittweisen Rückwärtsabarbeitung werden überhaupt keine Triggeraktivitäten ausgeführt.

Einschränkungen

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion CapL kürzer als normal ist (z. B. am Start von CapL mit der Roboterposition am Endpunkt), sind eventuell einige oder alle Triggerbedingungen sofort und an derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine „unvollständige Bewegung“ nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

Das Verhalten des CAP-Prozesses ist eventuell nicht definiert, wenn ein Fehler während der InstruktionenCapL oder CapC bei extrem kurzen TCP Bewegungen (< 1 mm) auftritt.

CapL kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
CapL
  [ToPoint ':='] < expression (IN) of robtarget >
  ['\' Id ':=' < expression (IN) of identno >] ','
  [Speed ':='] < expression (IN) of speeddata > ','
  [Cdata ':='] < persistent (PERS) of capdata >
  ['\' Movestart_timer ':=' < expression (IN) of num >] ','
  [Weavestart ':='] <persistent (PERS) of weavestartdata > ','
  [Weave ':='] < persistent (PERS) of capweavedata > ','
  [Zone ':='] < expression (IN) of zonedata >
  ['\' Inpos ':=' < expression (IN) of stoppointdata >] ','
  [Tool ':='] < persistent (PERS) of tooldata >
  ['\' WObj ':=' < persistent (PERS) of wobjdata >]
  ['\' Track ':=' < persistent (PERS) of captrackdata >]
  |[ '\' Corr]
  |[ '\' PreProcessTracking]
  ['\' Time ':=' < expression (IN) of num >]
  ['\' T1 ':=' < variable (VAR) of triggdata >]
  ['\' T2 ':=' < variable (VAR) of triggdata >]
  ['\' T3 ':=' < variable (VAR) of triggdata >]
  ['\' T4 ':=' < variable (VAR) of triggdata >]
  ['\' T5 ':=' < variable (VAR) of triggdata >]
  ['\' T6 ':=' < variable (VAR) of triggdata >]
  ['\' T7 ':=' < variable (VAR) of triggdata >]
  ['\' T8 ':=' < variable (VAR) of triggdata >]
  ['\' TLoad':=' < persistent (PERS) of loaddata >] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Lineare Bewegung	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Lineare Bewegung mit Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*
Definition der CAP-Daten	*capdata - CAP-Daten auf Seite 1653*
Definition von Pendelstartdaten	*weavestartdata - Pendelstartdaten auf Seite 1850*
Definition von Pendeldaten	*capweavedata - Pendeldaten für CAP auf Seite 1667*
Definition von Track-Daten	*captrackdata - CAP-Track-Daten auf Seite 1664*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*
Verwendung von optischen Sensoren für Nachverfolgung oder Suche.	*Operating manual - Tracking and searching with optical sensors*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.36 CapLATrSetup - Look-Ahead-Tracker einrichten

Verwendung

`CapLATrSetup` *(Look-Ahead-Tracker einrichten)* wird verwendet, um einen Sensortyp "Look-Ahead-Tracker" einzurichten, beispielsweise "Laser Tracker".

Die Sensorschnittstelle kommuniziert mit Hilfe des RTP1-Transportprotokolls über serielle Kanäle mit maximal zwei Sensoren. Die beiden Kanäle müssen *laser1:* und *swg:* genannt werden.

Grundlegendes Beispiel

SIO.cfg:

```
COM_TRP:
-Name "SCOUT:" -Type "RTP1"
-Name "digi-ip:" -Type "SOCKDEV" -PhyChannel "LAN1" -RemoteAdress
     "192.168.125.5"
```

RAPID-Code:

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
! Sensor calibration frame
PERS pose calibFrame := [[236.4,0.3,96.3],[1,0,0,0]];
! Trackdata
PERS captrackdata captrack1 := ["digi-ip:", [1,10,1,0,0,0,0,0]];

! Set up a Laser Tracker
CapLATrSetup “digi-ip:”,
     calibFrame\SensorFreq:=20\CorrFilter:=5\MaxBlind:=100\MaxIncCorr:=2;

! Request start of sensor measurements
WriteVar "digi-ip:", SensorOn, 1;

! Track using Cap
CapL p_fig1_l_1, v200, cd_event1, wsd_event, cwd_event, z20,
     tWeldGun\Track:=captrack1;

! Stop sensor
WriteVar "digi-ip:", SensorOn, 0;
```

Argumente

```
CapLATrSetup device CalibFrame CalibPos [\WarnMaxCorr] [\LogFile]
     [\LogSize] [\SensorFreq] [\IpolServoDelay] [\IpolCorrGain]
     [\ServoSensFactor] [\CorrFilter] [\IpolCorrFilter]
     [\ServoCorrFilter] [\ErrRampIn] [\ErrRampOut] [\CBAngle]
     [\MaxBlind] [\MaxIncCorr] [\CalibFrame2] [\CalibFrame3]
```

device

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Gerätename wie in sio.cfg definiert.

calibframe

Datentyp: `pose`

Koordinatensystem zur LATR-Kalibrierung (Position und Orientierung relativ zum vorgegebenen Werkzeug Tool0.)

CalibPos

Datentyp: `pose`

Offset der LATR-Kalibrierung. Anpassung des Sensor-Koordinatensystems, mit der die Origo des Koordinatensystems zur Bahnkorrektur in der Nähe der Ebene des bei der Korrektur verwendeten Werkzeug-Koordinatensystems platziert wird.

[\WarnMaxCorr]

Datentyp: `switch`

Wenn dieser Schalter vorhanden ist und der in den trackdata angegebene Grenzwert für die maximale Korrektur überschritten wurde, wird die Programmabarbeitung nicht unterbrochen. Es wird lediglich eine Warnung ausgegeben.

[\Logfile]

Datentyp: `string`

Name der Tracklog-Protokolldatei

[\LogSize]

Datentyp: `num`

Größe des Tracklog-Ringpuffers, das heißt die Anzahl der Sensormessungen, die während der Verfolgung gepuffert werden können.

Standard: 1000.

[\SensorFreq]

Datentyp: `num`

Definiert die Abtastfrequenz des verwendeten Sensors. M-Spot-90 hat z. B. eine Abtastfrequenz von 5 Hz.

Der höchste verfügbare Wert ist abhängig von der Kommunikationsverbindung und ihrer Geschwindigkeit. Wir empfehlen, keine Werte über 20 Hz zu verwenden.

Standard: 5 Hz.

[\IpolServoDelay]

Datentyp: `num`

Definiert die interne Zeitverzögerung der Robotersteuerung zwischen den Tasks IPOL und SERVO.

Standard: 74 ms.



Hinweis

Ändern Sie nicht den Standardwert!

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.36 CapLATrSetup - Look-Ahead-Tracker einrichten

Continuous Application Platform (CAP)
Fortsetzung

[\IpolCorrGain]

Datentyp: num

Definiert den Verstärkungsfaktor für die auf IPOL angewendete Korrektur.

Standard: 0.0.

Hinweis

Ändern Sie nicht den Standardwert!

[\ServoSensFactor]

Datentyp: num

Definiert die Anzahl der Servokorrekturen pro Sensorablesewert.

Standard: 0.

Hinweis

Ändern Sie nicht den Standardwert!

[\CorrFilter]

Datentyp: num

Definiert die Filterung für die berechnete Korrektur unter Verwendung des Mittelwerts über den Correction Filter-Werten.

Standard: 1.

Hinweis

Ändern Sie nicht den Standardwert!

[\IpolCorrFilter]

Datentyp: num

Definiert die Filterung für die IPOL-Korrektur unter Verwendung des Mittelwerts über den Bahnfilterwerten.

Standard: 1.

Hinweis

Ändern Sie nicht den Standardwert!

[\ServoCorrFilter]

Datentyp: num

Definiert die Filterung für die SERVO-Korrektur unter Verwendung des Mittelwerts über den Bahnservofilterwerten.

Standard: 1.

Hinweis

Ändern Sie nicht den Standardwert!

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\ErrRampIn]

Datentyp: `num`

Definiert die Anzahl der Sensorablesewerte, bei denen Ramp In durchgeführt wird, nachdem ein Fehler durch einen Sensorablesewert verursacht wurde.

Standard: 1.

[\ErrorRampOut]

Datentyp: `num`

Definiert die Anzahl der Sensorablesewerte, bei denen Ramp Out durchgeführt wird, nachdem ein Fehler durch einen Sensorablesewert verursacht wurde.

Standard: 1.

[\CBAngle]

Datentyp: `num`

Definiert den Winkel zwischen einem 3D-Sensorstrahl und der Z-Achse des Sensors.

Standard: 0.0.

[\MaxBlind]

Datentyp: `num`

Maximaler Abstand, um den sich der TCP verschieben kann, unter der Annahme, dass die neueste Korrektur noch gültig ist.

Zu Beginn der Verfolgung wird der `MaxBlind`-Abstand durch die Optik vor dem Sensor automatisch erhöht.

Standard: keine Begrenzung.

[\MaxIncCorr]

Datentyp: `num`

Maximale inkrementelle Korrektur erlaubt.

Wenn die inkrementelle TCP-Korrektur größer als `\MaxIncCorr` ist und `\WarnMaxCorr` festgelegt wurde, setzt der Roboter zwar seine Bahn fort, aber die übernommene inkrementelle Korrektur überschreitet `\MaxIncCorr` nicht. Wenn `\WarnMaxCorr` nicht festgesetzt wurde, wird ein Verfolgungsfehler gemeldet und die Programmabarbeitung angehalten.

Standard: 5 mm.

[\CalibFrame2]

Datentyp: `pose`

Alternative Nummer des Koordinatensystems zur LATR-Kalibrierung 2 (Position und Orientierung relativ zum vorgegebenen Werkzeug `tool0`).

[\CalibFrame3]

Datentyp: `pose`

Alternative Nummer des Koordinatensystems zur LATR-Kalibrierung 3 (Position und Orientierung relativ zum vorgegebenen Werkzeug `tool0`).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.36 CapLATrSetup - Look-Ahead-Tracker einrichten

Continuous Application Platform (CAP)
Fortsetzung

Syntax

```
CapLATrSetup
  [device ':='] < expression (IN) of string> ','
  [CalibFrame ':='] < persistent (PERS) of pose > ','
  [CalibPos ':='] < persistent (PERS) of pos >
  [\WarnMaxCorr]
  [\LogFile ':=' < expression (IN) of string >]
  [\LogSize ':=' < expression (IN) of num >]
  [\SensorFreq ':=' < expression (IN) of num >]
  [\IpolServoDelay ':=' < expression (IN) of num >]
  [\IpolCorrGain ':=' < expression (IN) of num >]
  [\ServoSensFactor ':=' < expression (IN) of num >]
  [\CorrFilter ':=' < expression (IN) of num >]
  [\IpolCorrFilter ':=' < expression (IN) of num >]
  [\ServoCorrFilter ':=' < expression (IN) of num >]
  [\ErrRampIn ':=' < expression (IN) of num >]
  [\ErrRampOut ':=' < expression (IN) of num >]
  [\CBAngle ':=' < expression (IN) of num >]
  [\MaxBlind ':=' < expression (IN) of num >]
  [\MaxIncCorr ':=' < expression (IN) of num >]
  [\CalibFrame2 ':=' < persistent (PERS) of pose >]
  [\CalibFrame3 ':=' < persistent (PERS) of pose >] ';'
```

Weitere Informationen

Information über	Siehe
Sensor Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.37 CapNoProcess - CAP ohne Prozess starten

Verwendung

`CapNoProcess` wird verwendet, um CAP für einen bestimmten Abstand ohne Prozess auszuführen.

Mit `CapNoProcess` ist es möglich, CAP anzuweisen, eine bestimmte Entfernung (in mm) ohne Prozess auszuführen. Das ist sinnvoll, wenn es einen behebbaren Prozessfehler gab, der es in irgendeiner Art und Weise unmöglich macht, das Verfahren bei der Fehlerstelle neu zu starten.

Am Anfang und am Ende des übersprungenen Abstands wird die Komponente der Rückwärtsbewegung auf die Bahn (`restart_dist` in `capdata`) unterdrückt.

Am Ende des übersprungenen Abstands wird ein Fehler mit errno CAP_NOPROC_END erzeugt.

Grundlegendes Beispiel

```
VAR num skip_dist := 0.0;
VAR bool cap_skip := FALSE;

PROC main()
  ......
  skip_dist := 25.0;
  CapL p_fig3_l_1, v500, cd, wsd, cwd, fine, tWeldGun;
  ......
  skip_dist := 15.0;
  CapL p_fig3_l_3, v500, cd, wsd, cwd, fine, tWeldGun;
  ......

  ERROR
    StorePath;
    TEST ERRNO
    CASE CAP_NOPROC_END:
      IF cap_skip THEN
        ! This is the end of the skip distance
        cap_skip := FALSE;
      ENDIF
    CASE CAP_MAIN_ERR:
      IF skip_dist > 0.0 THEN
        ! This is the start of the skip distance
        CapNoProcess skip_dist;
        cap_skip := TRUE;
      ENDIF
    DEFAULT:
    ENDTEST
    RestoPath;
    StartMoveRetry;
ENDPROC
ENDMODULE
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.37 CapNoProcess - CAP ohne Prozess starten

Continuous Application Platform (CAP)
Fortsetzung

In diesem Beispiel folgt auf den behebbaren Fehler CAP_MAIN_ERR eine Bewegung ohne Prozess von 25 mm (mit 10 mm/s) bei der ersten `CapL`-Instruktion und von 15 mm bei der zweiten. Am Ende dieses Abstands wird CAP_NOPROC_END erzeugt und der Prozess neu gestartet.

Argumente

```
CapNoProcess skip_distance
```

skip_distance

Abstand in mm

Datentyp: `num`

`CapNoProcess` hat eine `num`-Variable als Eingangsparameter, die den übersprungenen Abstand in mm definiert.

Einschränkungen

Die Geschwindigkeit des TCP während des Überspringens wird mit 10 mm/s vorgegeben. Der kürzeste Sprungentfernung wird mit 10 mm vorgegeben.

In synchronisierten MultiMove-Systemen ist die kürzeste Entfernung aller Sprungabstände, die für die verschiedenen synchronisierten Prozessroboter definiert sind, die tatsächliche Entfernung.

Wenn der Sprungabstand länger ist als die Entfernung von der aktuellen TCP-Position bis zum Ende der aktuellen Sequenz der CAP-Instruktionen, geschieht nichts Besonderes: die RAPID-Abarbeitung geht normal weiter, ohne den Roboter zu stoppen.

Syntax

```
CapNoProcess
  [skip_dist ':='] < variable (IN) of num >';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instruktion `InitSuperv`	*InitSuperv - Zurücksetzen der gesamten CAP-Überwachung auf Seite 284*
Instruktion `SetupSuperv`	*SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP auf Seite 725*
Instruktion `RemoveSuperv`	*RemoveSuperv Bedingung für ein Signal entfernen auf Seite 607*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.38 CapRefresh - CAP-Daten aktualisieren

Verwendung

`CapRefresh` wird verwendet, um den CAP-Prozess anzuweisen, seine Prozessdaten zu aktualisieren. Das kann zum Beispiel zum Abstimmen der CAP-Prozessparameter während der Programmabarbeitung verwendet werden.

Grundlegendes Beispiel

```
PROC PulseSpeed()
  ! Setup a 1 Hz timer interrupt
  CONNECT intno1 WITH TuneTrp;
  ITimer 1, intno1;
  CapL p1, v100, cdata, wstartdata, wdata, fine, gun1;
  IDelete intno1;
ENDPROC

TRAP TuneTrp
  ! Modify the main speed component of active cdata
  IF HighValueFlag = TRUE THEN
    cdata.speed_data.start := 10;
    HighValueFlag := FALSE;
  ELSE
    cdata.speed_data.start := 15;
    HighValueFlag := TRUE;
  ENDIF
  ! Order the process control to refresh process parameters
  CapRefresh;
ENDTRAP
```

In diesem Beispiel wird die Geschwindigkeit zwischen 10 und 15 mm/s mit einer Rate von 1 Hz umgeschaltet.

Argumente

```
CapRefresh [\MainSpeed] [\MainWeave] [\StartWeave] [\RestartDist]
```

Ohne optionales Argument werden die CAP-Daten `capdata`, `capweavedata`, `weavestartdata`, `captrackdata` und `movestarttimer`, falls vorhanden, aus der RAPID-Variable PERSISTENT gelesen, die in der aktuell aktiven CAP-Instruktion spezifiziert wurde.

[\MainSpeed]

Datentyp: `switch`

Wenn dieser Schalter vorhanden ist, liest CAP wieder die Komponente `capdata.speed_data.main` der gerade aktiven CAP-Instruktion.

[\MainWeave]

Datentyp: `switch`

Wenn dieser Schalter vorhanden ist, liest GAP wieder die Komponenten `capweavedata.width`, `capweavedata. length`, `capweavedata.bias` und `capweavedata.active` der gerade aktiven CAP-Instruktion.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.38 CapRefresh - CAP-Daten aktualisieren

Continuous Application Platform (CAP)
Fortsetzung

[\StartWeave]

Datentyp: `bool`

Wenn dieser Schalter vorhanden ist, wird der CAP seinen Wert anstelle von `weavestartdata.active` der gerade aktiven CAP-Instruktion verwenden. Die Daten der gerade aktiven CAP-Instruktion bleiben davon unberührt.

[\RestartDist]

Datentyp: `num`

Wenn dieser Schalter vorhanden ist, wird der CAP seinen Wert anstelle von `capdata.restart_dist` der gerade aktiven CAP-Instruktion verwenden. Die Daten der gerade aktiven CAP-Instruktion bleiben davon unberührt.

Syntax

```
CapRefresh
  ['\' MainSpeed]
  ['\' MainWeave]
  ['\' Startweave ':=' < expression (IN) of bool >]
  ['\' RestartDist ':=' < expression (IN) of num >] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.39 CAPSetStopMode - Stellen Sie den Stopp-Modus für Execution Error ein

Verwendung

`CAPSetStopMode` legt den Stopp-Modus fest, der verwendet werden soll, wenn die Roboterbewegung aufgrund eines Prozessfehlers gestoppt wird. Der Standard-Stopp-Modus ist SMOOTH_STOP_ON_PATH.

Grundlegendes Beispiel

```
CAPSetStopMode QUICK_STOP_ON_PATH;
```

Argumente

```
CAPSetStopMode StopMode;
```

StopMode

Datentyp: `capstopmode`

Stopp-Modus, siehe *capstopmode - Definiert die Stopp-Modi für CAP auf Seite 1663*.

Syntax

```
CAPSetStopMode
  [StopMode ':='] <variable (VAR) of capstopmode> ';'
```

Weitere Informationen

Information über	Siehe
Datentyp capstopmode	*capstopmode - Definiert die Stopp-Modi für CAP auf Seite 1663*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.40 CapWeaveSync - Signale und Ebenen für die Synchronisierung der Pendelbewegung einrichten.

Verwendung

`CapWeaveSync` wird verwendet, um die Pendelsynchronisierungssignale ohne Sensoren festzulegen. Die E/A-Signale müssen in EIO.cfg definiert werden.

Grundlegendes Beispiel

RAPID-Programm:

```
PROC main()
  ...
  CapWeaveSync \DoLeft:=do_sync_left \LevelLeft:=80
       \DoRight:=do_sync_right \LevelRight:=80;
  ...
ENDPROC
```

In diesem Beispiel sind die Signale `do_sync_left and do_sync_right` auf einen Pendelebene von 80% eingestellt.

Die `CapWeaveSync`-Instruktion darf nur einmal ausgeführt werden z.B. vor dem Anlaufen.

Argumente

```
CapWeaveSync [\Reset] [\DoLeft] [\LevelLeft] [\DoRight]
     [\LevelRight]
```

[\Reset]

Datentyp: `switch`

Pendelsynchronisierungsdaten löschen.

[\DoLeft]

Datentyp: `signaldo`

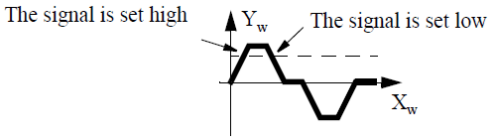
Das digitale Ausgangssignal für die Synchronisation der Pendelbewegung auf der linken Seite des Pendelzyklus.

[\LevelLeft]

Datentyp: `num`

Die Koordinierungsposition auf der linken Seite des Pendelbewegungsmusters. Sie wird als Prozentsatz der Breite links von der Pendelmitte angegeben. Wenn die Pendelbewegung über diesen Punkt hinausgeht, wird automatisch ein hohes digitales Ausgangssignal gesetzt (wenn das Signal definiert ist).

Die Koordinationsart ist für die Nahtverfolgung mittels des Through-the-Arc-Trackers gedacht.



xx1200000176

Fortsetzung auf nächster Seite

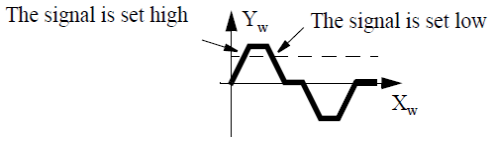
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\LevelLeft]

Datentyp: `num`

Die Koordinierungsposition auf der linken Seite des Pendelbewegungsmusters. Sie wird als Prozentsatz der Breite links von der Pendelmitte angegeben. Wenn die Pendelbewegung über diesen Punkt hinausgeht, wird automatisch ein hohes digitales Ausgangssignal gesetzt (wenn das Signal definiert ist).

Die Koordinationsart ist für die Nahtverfolgung mittels des Through-the-Arc-Trackers gedacht.



xx1200000176

[\DoRight]

Datentyp: `signaldo`

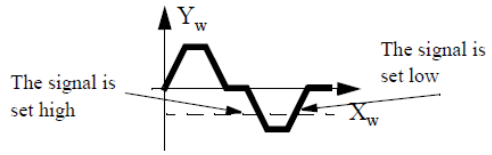
Das digitale Ausgangssignal für die Synchronisation der Pendelbewegung auf der rechten Seite des Pendelzyklus.

[\LevelRight]

Datentyp: `num`

Die Koordinierungsposition auf der rechten Seite des Pendelbewegungsmusters. Sie wird als Prozentsatz der Breite rechts von der Pendelmitte angegeben. Wenn die Pendelbewegung über diesen Punkt hinausgeht, wird automatisch ein hohes digitales Ausgangssignal gesetzt (wenn das Signal definiert ist).

Die Koordinationsart ist für die Nahtverfolgung mittels des Through-the-Arc-Trackers gedacht.



xx1200000177

Programmabarbeitung

Die definierten Signale werden überprüft und gesetzt, wenn der Sensor nicht aktiv ist.

Einschränkungen

Die Signale müssen in EIO.cfg definiert sein.

Es kann entweder nur eine Ebene oder das entsprechendes Signal verwendet werden. Das führt nicht zu Fehlern beim Laden der RAPID-Datei, aber es führt zu RAPID-Laufzeitfehlern der Instruktion `CapWeaveSynch`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.40 CapWeaveSync - Signale und Ebenen für die Synchronisierung der Pendelbewegung einrichten.

Continuous Application Platform (CAP)

Fortsetzung

Syntax

```
CapWeaveSync
  ['\' Reset]
  [DoLeft ':=' < expression (IN) of signaldo >]
  [LevelLeft ':=' < expression (IN) of num >]
  [DoRight ':=' < expression (IN) of signaldo >]
  [LevelRight ':=' < expression (IN) of num >] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
`capweavedata` **Datentyp**	*capweavedata - Pendeldaten für CAP auf Seite 1667*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.41 CheckProgRef - Überprüfen von Programmverweisen

Verwendung

`CheckProgRef` wird verwendet, um zu einem beliebigen Zeitpunkt der Abarbeitung zu überprüfen, ob nicht aufgelöste Verweise vorhanden sind.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CheckProgRef`:

Beispiel 1:

```
Load \Dynamic, diskhome \File:="PART_B.MOD" \CheckRef;
Unload "PART_A.MOD";
CheckProgRef;
```

In diesem Fall enthält das Programm das Modul `PART_A.MOD`. Das neue Modul `PART_B.MOD` wird geladen, das überprüft, ob alle Verweise ordnungsgemäß sind. Anschließend wird `PART_A.MOD` entladen. Um nach dem Entladen eine Überprüfung auf nicht aufgelöste Verweise durchzuführen, wird `CheckProgRef` ausgeführt.

Programmabarbeitung

Die Programmabarbeitung erzwingt eine neue Verknüpfung der Programmtasks und überprüft, ob nicht aufgelöste Verweise vorhanden sind.

Wenn während `CheckProgRef` ein Fehler auftritt, wirkt sich dies nicht auf das Programm aus. Es wird lediglich gemeldet, dass in der Programmtask ein nicht aufgelöster Verweis vorhanden ist. Verwenden Sie daher `CheckProgRef` sofort nach dem Ändern der Anzahl der Module in der Programmtask (beim Laden oder Entladen), um ermitteln zu können, durch welches Modul der Verknüpfungsfehler verursacht wurde.

Diese Instruktion kann auch anstelle des optionalen Arguments `\CheckRef` in der Instruktion `Load` oder `WaitLoad` verwendet werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_LINKREF`	Die Programmtask enthält nicht aufgelöste Verweise.

Syntax

```
CheckProgRef';'
```

Weitere Informationen

Information über	Siehe
Laden eines Programmmoduls	*Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346*
Entladen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.41 CheckProgRef - Überprüfen von Programmverweisen

RobotWare - OS

Fortsetzung

Information über	Siehe
Starten des Ladens eines Programmmoduls	*StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802*
Beenden des Ladens eines Programmmoduls	*WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.42 CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn

Verwendung

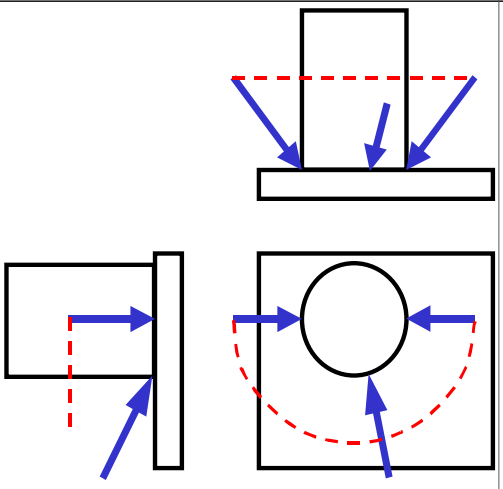
`CirPathMode` (*Circle Path Mode*) ermöglicht die Wahl verschiedener Modi zur Umorientierung des Werkzeugs bei kreisförmigen Bewegungen.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

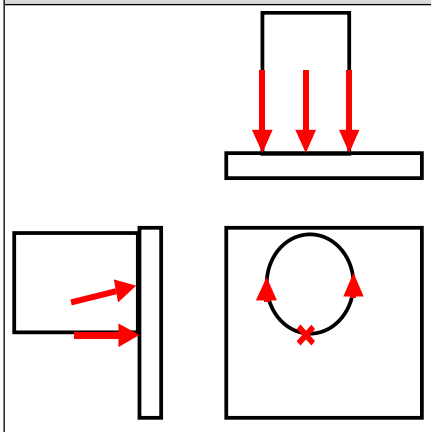
Beschreibung

PathFrame

Die Abbildung in der Tabelle zeigt die Umorientierung des Werkzeugs für den Standardmodus `\PathFrame`.

Abbildung	Beschreibung
 xx0500002152	Die Pfeile zeigen das Werkzeug von der Mitte des Handgelenks zum Werkzeugarbeitspunkt für die programmierten Punkte. Die Bahn für die Handgelenkmitte wird gepunktet dargestellt. Der Modus `\PathFrame` erleichtert es, denselben Werkzeugwinkel um den Zylinder zu erzielen. Das Handgelenk des Roboters durchläuft nicht die im `CirPoint` programmierte Orientierung.

Die Abbildung in der Tabelle zeigt die Verwendung des Standardmodus `\PathFrame` mit fester Werkzeugorientierung.

Abbildung	Beschreibung
 xx0500002153	Diese Abbildung zeigt die erzielte Orientierung des Werkzeugs in der Mitte des Kreises mit Hilfe eines Neigungswerkzeugs und im Modus `\PathFrame`. Vgl. mit nachstehender Abbildung, in der der Modus `\ObjectFrame` verwendet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

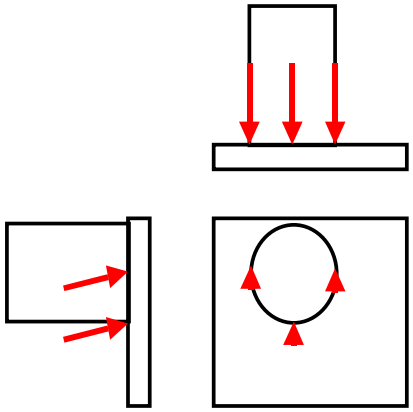
1.42 CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn

RobotWare - OS

Fortsetzung

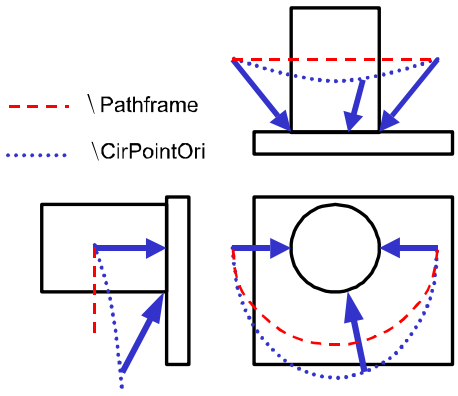
ObjectFrame

Die Abbildung in der Tabelle zeigt die Verwendung des modifizierten Modus `\ObjectFrame` mit fester Werkzeugorientierung.

Abbildung	Beschreibung
 xx0500002151	Diese Abbildung zeigt die erzielte Orientierung des Werkzeugs in der Mitte des Kreises mit Hilfe eines Neigungswerkzeugs und im Modus `\ObjectFrame`. Dieser Modus ergibt eine lineare Umorientierung des Werkzeugs wie für `MoveL`. Das Handgelenk des Roboters durchläuft nicht die programmierte Orientierung im `CirPoint`. Vergleichen Sie dies mit obiger Abbildung, in der der Modus `\PathFrame` verwendet wird.

CirPointOri

Die Abbildung in folgender Tabelle zeigt die unterschiedliche Umorientierung des Werkzeugs für den Standardmodus `\PathFrame` und den modifizierten Modus `\CirPointOri`.

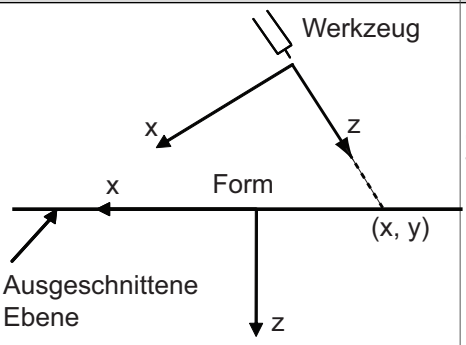
Abbildung	Beschreibung
 xx0500002150	Die Pfeile zeigen das Werkzeug von der Mitte des Handgelenks zum Werkzeugarbeitspunkt für die programmierten Punkte. Die unterschiedlichen Bahnen für die Handgelenkmitte werden gestrichelt dargestellt. Im Modus `\CirPointOri` durchläuft das Handgelenk des Roboters die programmierte Orientierung in `CirPoint`. Die Bahn ist für xyz immer dieselbe, aber die Orientierung ist unterschiedlich.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wrist45 / Wrist46 / Wrist56

Die Abbildung in folgender Tabelle zeigt die Koordinatensysteme beim Schneiden einer Form mit den Achsen 4 und 5.

Abbildung	Beschreibung
	Es wird angenommen, dass der Schneidstrahl an der z-Achse des Werkzeugs ausgerichtet ist. Das Koordinatensystem der ausgeschnittene Ebene wird durch die Startposition des Roboters definiert, wenn die `MoveC`-Instruktion abgearbeitet wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `CirPathMode`:

Beispiel 1:

```
CirPathMode \PathFrame;
```

Der Standardmodus für die Werkzeugumorientierung im tatsächlichen Bahn-Koordinatensystem vom Startpunkt bis zum Punkt `ToPoint` bei allen erfolgreichen kreisförmigen Bewegungen. Dies ist der Standard im System.

Beispiel 2:

```
CirPathMode \ObjectFrame;
```

Modifizierter Modus für die Werkzeugumorientierung im tatsächlichen Objekt-Koordinatensystem vom Startpunkt bis zum Punkt `ToPoint` bei allen erfolgreichen kreisförmigen Bewegungen.

Beispiel 3

```
CirPathMode \CirPointOri;
```

Modifizierter Modus für die Werkzeugneuorientierung vom Startpunkt über die programmierte Orientierung `CirPoint` bis zum Punkt `ToPoint` bei allen erfolgreichen kreisförmigen Bewegungen.

Beispiel 4:

```
CirPathMode \Wrist45;
```

Modifizierter Modus, damit die Projektion der Z-Achse des Werkzeugs auf die ausgeschnittene Ebene der programmierten Kreisbewegungsreihenfolge entspricht. Es werden nur die Handgelenkachsen 4 und 5 verwendet. Dieser Modus sollte nur beim Schneiden von dünnen Objekten verwendet werden.

Beispiel 5:

```
CirPathMode \Wrist46;
```

Modifizierter Modus, damit die Projektion der Z-Achse des Werkzeugs auf die ausgeschnittene Ebene der programmierten Kreisbewegungsreihenfolge entspricht. Es werden nur die Handgelenkachsen 4 und 6 verwendet. Dieser Modus sollte nur bei dünnen Objekten verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.42 CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn

RobotWare - OS

Fortsetzung

Beispiel 6:

```
CirPathMode \Wrist56;
```

Modifizierter Modus, damit die Projektion der Z-Achse des Werkzeugs auf die ausgeschnittene Ebene der programmierten Kreisbewegungsreihenfolge entspricht. Es werden nur die Handgelenkachsen 5 und 6 verwendet. Dieser Modus sollte nur bei dünnen Objekten verwendet werden.

Argumente

```
CirPathMode [\PathFrame] | [\ObjectFrame] | [\CirPointOri] |
     [\Wrist45] | [\Wrist46] | [\Wrist56]
```

`[\PathFrame]`

Datentyp: `switch`

Bei der kreisförmigen Bewegung erfolgt die Werkzeugumorientierung kontinuierlich vom Startpunkt der Orientierung zum Punkt `ToPoint` im tatsächlichen Bahn-Koordinatensystem. Dies ist der Standardmodus im System.

> **Hinweis**
>
> Das Einsetzen von `CirPathMode` ohne Schalter gibt dasselbe Ergebnis wie bei `CirPathMode \PathFrame`.

`[\ObjectFrame]`

Datentyp: `switch`

Bei der kreisförmigen Bewegung erfolgt die Werkzeugumorientierung kontinuierlich vom Startpunkt der Orientierung zum Punkt `ToPoint` im tatsächlichen Objekt-Koordinatensystem.

`[\CirPointOri]`

Datentyp: `switch`

Bei der kreisförmigen Bewegung erfolgt die Werkzeugumorientierung kontinuierlich vom Startpunkt der Orientierung zur programmierten Orientierung `CirPoint` und weiter zur Orientierung `ToPoint`.

`[\Wrist45]`

Datentyp: `switch`

Der Roboter bewegt die Achsen 4 und 5 so, dass die Projektion der Z-Achse des Werkzeugs auf die ausgeschnittene Ebene der programmierten Kreisbewegungsreihenfolge entspricht. Dieser Modus sollte nur bei dünnen Objekten verwendet werden, da nur 2 Handgelenk-Achsen eingesetzt werden, die zwar für eine größere Genauigkeit sorgen, sich aber schlechter steuern lassen.

> **Hinweis**
>
> Dieser Schalter erfordert die RobotWare-Option *WristMove*.

`[\Wrist46]`

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Roboter bewegt die Achsen 4 und 6 so, dass die Projektion der Z-Achse des Werkzeugs auf die ausgeschnittene Ebene der programmierten Kreisbewegungsreihenfolge entspricht. Dieser Modus sollte nur bei dünnen Objekten verwendet werden, da nur 2 Handgelenk-Achsen eingesetzt werden, die zwar für eine größere Genauigkeit sorgen, sich aber schlechter steuern lassen.

Hinweis

Dieser Schalter erfordert die RobotWare-Option *WristMove*.

`[\Wrist56]`

Datentyp: `switch`

Der Roboter bewegt die Achsen 5 und 6 so, dass die Projektion der Z-Achse des Werkzeugs auf die ausgeschnittene Ebene der programmierten Kreisbewegungsreihenfolge entspricht. Dieser Modus sollte nur bei dünnen Objekten verwendet werden, da nur 2 Handgelenk-Achsen eingesetzt werden, die zwar für eine größere Genauigkeit sorgen, sich aber schlechter steuern lassen.

Hinweis

Dieser Schalter erfordert die RobotWare-Option *WristMove*.

Programmabarbeitung

Der angegebene Modus für die kreisförmige Werkzeugumorientierung gilt für die nächste abgearbeitete kreisförmige Bewegung (`MoveC, SearchC, TriggC, MoveCDO, MoveCSync, ArcC, PaintC` usw.) und gilt so lange, bis eine neue`CirPathMode` Instruktion abgearbeitet wird.

Der Standardmodus für kreisförmige Umorientierung (`CirPathMode \PathFrame`) wird automatisch gesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.42 CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn

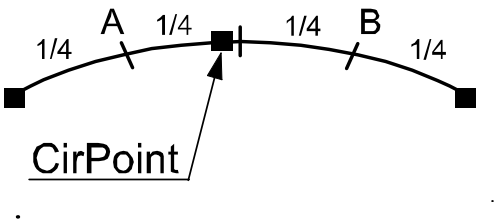
RobotWare - OS

Fortsetzung

Einschränkungen

Die Instruktion beeinflusst nur kreisförmige Bewegungen.

Bei Verwendung des Modus `\CirPointOri` muss `CirPoint` zwischen den Punkten **A** und **B** liegen (wie in nachstehender Abbildung gezeigt), damit die Kreisbewegung durch die programmierte Orientierung in `CirPoint` verläuft.



xx0500002149

`\Wrist45`, Die Modi `\Wrist46` und `\Wrist56` sollten nur bei dünnen Objekten verwendet werden, da beim Einsatz von nur zwei Handgelenk-Achsen die Steuerung des Werkzeugwinkels verloren geht. Koordinierte Bewegungen sind nicht möglich, da die Hauptachse gesperrt ist.

Wenn im Handgelenksingularitätsbereich die Instruktion `SingArea\Wrist` abgearbeitet wurde, ist die Instruktion `CirPathMode` ohne Wirkung, da das System dann einen anderen Modus der Werkzeugumorientierung für kreisförmige Bewegungen wählt (Achseninterpolation).

Syntax

```
CirPathMode
  ['\'PathFrame]
  | ['\'ObjectFrame]
  | ['\'CirPointOri]
  | ['\'Wrist45]
  | ['\'Wrist46]
  | ['\'Wrist56] ';'
```

Weitere Informationen

Information über	Siehe
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Instruktion für kreisförmige Bewegung	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Definieren der Interpolation in der Nähe singulärer Punkte	*SingArea - Definieren der Interpolation in der Nähe singulärer Punkte auf Seite 736*
Interpolation	*Technisches Referenzhandbuch - RAPID Overview*
Handgelenkbewegungen	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *WristMove*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.43 Clear - Löschen des Werts

Verwendung

`Clear` wird verwendet., um eine numerische Variable oder Persistente zu löschen, d. h. auf 0 zu setzen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Clear`:

Beispiel 1:

```
Clear reg1;
```

`Reg1` wird gelöscht, d. h. `reg1:=0`.

Beispiel 2:

```
CVAR dnum mydnum:=5;
Clear mydnum;
```

`mydnum` wird gelöscht, d. h. `mydnum:=0`.

Argumente

```
Clear Name | Dname
```

`Name`

Datentyp: `num`

Der Name der zu löschenden Variablen oder Persistenten.

`Dname`

Datentyp: `dnum`

Der Name der zu löschenden Variablen oder Persistenten.

Syntax

```
Clear
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname ':=' ] < var or pers (INOUT) of dnum > ';'
```

Weitere Informationen

Information über	Siehe
Inkrementieren der Variablen um 1	*Incr - Inkrementieren um 1 auf Seite 260*
Dekrementieren der Variable um 1	*Decr - Dekrementieren um 1 auf Seite 189*
Addieren eines Werts zu einer Variablen	*Add – Addiert einen numerischen Wert auf Seite 32*
Ändern von Daten über einen beliebigen Ausdruck	*":=" - Zuweisen eines Werts auf Seite 42*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.44 ClearIOBuff - Löschen des Eingangspuffers eines seriellen Kanals

Verwendung

`ClearIOBuff` (*Clear I/O Buffer*) wird verwendet, um den Eingangspuffer eines seriellen Kanals zu löschen. Alle gepufferten Zeichen aus dem seriellen Eingangskanal werden entfernt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ClearIOBuff`:

Beispiel 1:

```
VAR iodev channel1;
...
Open "com1:", channel1 \Bin;
ClearIOBuff channel1;
WaitTime 0.1;
```

Der Eingangspuffer für den seriellen Kanal `channel1` wird gelöscht. Die Wartezeit gewährt genügend Zeit zum Abschließen des Löschvorgangs.

Argumente

```
ClearIOBuff IODevice
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) des seriellen Kanals, dessen Eingangspuffer gelöscht werden soll.

Programmabarbeitung

Alle gepufferten Zeichen aus dem seriellen Eingangskanal werden entfernt. Die nächsten Leseinstruktionen warten auf neue Eingaben vom Kanal.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jeder geöffnete serielle Kanal im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Einschränkungen

Diese Instruktion kann nur für serielle Kanäle verwendet werden. Es wird nicht auf die Bestätigung für den Abschluss der Operation gewartet. Nach der Instruktion wird eine Wartezeit von 0,1 empfohlen, damit für die Operation in jeder Anwendung genügend Zeit zur Verfügung steht.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_FILEACC`	Die Instruktion wird für eine Datei verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
ClearIOBuff
  [IODevice ':='] <variable (VAR) of iodev>';'
```

Weitere Informationen

Information über	Siehe
Öffnen eines seriellen Kanals	*Technisches Referenzhandbuch - RAPID Overview*
File and Serial Channel Handling	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.45 ClearPath - Löschen der aktuellen Bahn

Verwendung

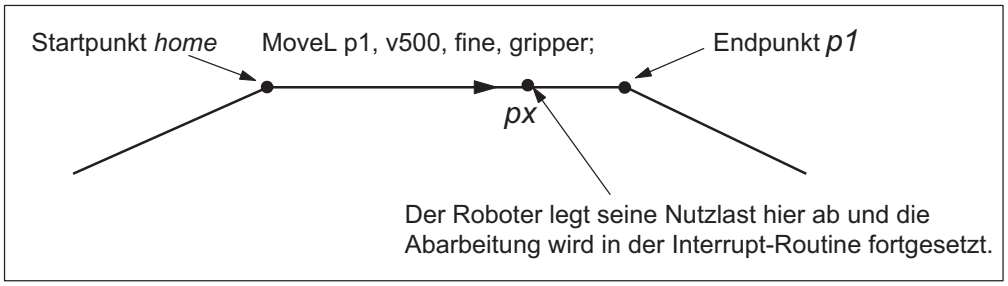
`ClearPath` (*Clear Path*) löscht die gesamte Bewegungsbahn auf der aktuellen Bewegungsbahnebene (Basisebene oder `StorePath`-Ebene).

Mit Bewegungsbahn werden alle Bewegungsfolgen aller Bewegungsinstruktionen bezeichnet, die in RAPID ausgeführt wurden, jedoch zum Abarbeitungszeitpunkt von `ClearPath` noch nicht vom Roboter durchgeführt wurden.

Der Roboter muss sich in einer Stopppunktposition befinden oder durch `StopMove` angehalten worden sein, bevor die Instruktion `ClearPath` abgearbeitet werden kann.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ClearPath`:



xx0500002154

Im folgenden Programmbeispiel bewegt sich der Roboter von der Position `home` zur Position `p1`. Am Punkt `px` gibt das Signal `di1` an, dass die Nutzlast abgelegt wurde. Die Abarbeitung wird in der Interrupt-Routine `gohome` fortgesetzt. Der Roboter stoppt seine Bewegung (beginnt mit dem Abbremsen) an Punkt `px`, die Bahn wird gelöscht und der Roboter bewegt sich an Position `home`. Der Fehler wird an die aufrufende Routine `minicycle` übergeben und der vollständige benutzerdefinierte Programmzyklus `proc1 ... proc2` wird noch einmal von Anfang an abgearbeitet.

Beispiel 1:

```
VAR intnum drop_payload;
VAR errnum ERR_DROP_LOAD := -1;

PROC minicycle()
  BookErrNo ERR_DROP_LOAD;
  proc1;
  ...
  ERROR (ERR_DROP_LOAD)
    ! Restart the interupted movement on motion base path level
    StartMove;
    RETRY;
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
PROC proc1()
  ...
  proc2;
  ...
ENDPROC

PROC proc2()
  CONNECT drop_payload WITH gohome;
  ISignalDI \Single, di1, 1, drop_payload;
  MoveL p1, v500, fine, gripper;
  ...........
  IDelete drop_payload;
ENDPROC

TRAP gohome
  StopMove;
  ClearPath;
  IDelete drop_payload;
  StorePath;
  MoveL home, v500, fine, gripper;
  RestoPath;
  RAISE ERR_DROP_LOAD;
  ERROR
    RAISE;
ENDTRAP
```

Wenn jedoch dasselbe Programm ohne`StopMove` **und** `ClearPath` **in der Trap-Routine** `gohome` **ausgeführt wird, so fährt der Roboter mit Position** `p1` **fort, bevor er in Position** `home` **zurückkehrt.**

Einschränkungen

Grundlegende Beispiele für die Instruktion `ClearPath` **sind unten aufgeführt.**

Beispiel 1 - Einschränkung

```
VAR intnum int_move_stop;
...
PROC test_move_stop()
  CONNECT int_move_stop WITH trap_move_stop;
  ISignalDI di1, 1, int_move_stop;
  MoveJ p10, v200, z20, gripper;
  MoveL p20, v200, z20, gripper;
ENDPROC

TRAP trap_move_stop
  StopMove;
  ClearPath;
  StorePath;
  MoveJ p10, v200, z20, gripper;
  RestoPath;
  StartMove;
ENDTRAP
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dies ist ein Beispiel für eine Einschränkung mit `ClearPath`. Während der Roboterbewegung nach `p10` und `p20` wird die Bewegung angehalten und die Bewegungsbahn gelöscht, doch es wird nichts unternommen, um die aktive Instruktion `MoveJ p10` oder `MoveL p20` in `PROC test_move_stop` abzubrechen. Daher wird die gegenwärtige Bewegung unterbrochen und der Roboter bewegt sich in `TRAP trap_move_stop` nach `p10`, doch es erfolgt in `PROC test_move_stop` keine weitere Bewegung nach `p10` oder `p20`. Die Programmabarbeitung bleibt hängen.

Dieses Problem lässt sich mit der Fehlerbehandlung und langem Sprung wie in Beispiel 2 oder mit einem mit der Instruktion `ProcerrRecovery` asynchron ausgelösten Fehler lösen.

Beispiel 2 - Keine Einschränkungen

```
VAR intnum int_move_stop;
VAR errnum err_move_stop := -1;
...
PROC test_move_stop()
  BookErrNo err_move_stop;
  CONNECT int_move_stop WITH trap_move_stop;
  ISignalDI di1, 1, int_move_stop;
  MoveJ p10, v200, z20, gripper;
  MoveL p20, v200, z20, gripper;
  ERROR (err_move_stop)
    StopMove;
    ClearPath;
    StorePath;
    MoveJ p10, v200, z20, gripper;
    RestoPath;
    ! Restart the interupted movement on motion base path level
    StartMove;
    RETRY;
ENDPROC

TRAP trap_move_stop
  RAISE err_move_stop;
  ERROR
    RAISE;
ENDTRAP
```

Dies ist ein Beispiel für die Verwendung der Fehlerbehandlung mit langem Sprung gemeinsam mit `ClearPath` ohne jede Einschränkung. Während der Roboterbewegung zu `p10` und `p20` wird die Bewegung gestoppt. Die Bewegungsbahn wird gelöscht, und aufgrund der abarbeitungsebenenübergreifenden Fehlerbehandlung wird die aktive Instruktion `MoveJ p10` oder `MoveL p20` abgebrochen. Daher wird die gegenwärtige Bewegung unterbrochen und der Roboter bewegt sich in `ERROR handler` nach `p10`, und die unterbrochene Instruktion `MoveJ p10` oder `MoveL p20` in `PROC test_move_stop` wird ein weiteres Mal ausgeführt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
ClearPath ';'
```

Weitere Informationen

Information über	Siehe
Stoppen von Roboterbewegungen	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview* *Technical reference manual - RAPID kernel*
Asynchron ausgelöste Fehler	*ProcerrRecovery - Generieren und Beheben von Bewegungsfehlern auf Seite 547*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.46 ClearRawBytes - Löschen des Inhalts von rawbytes-Daten

Verwendung

`ClearRawBytes` wird verwendet, um den vollständigen Inhalt einer `rawbytes`-Variablen auf 0 zu setzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ClearRawBytes`:

Beispiel 1:

```
VAR rawbytes raw_data;
VAR num integer := 8
VAR num float := 13.4;

PackRawBytes integer, raw_data, 1 \IntX := DINT;
PackRawBytes float, raw_data, (RawBytesLen(raw_data)+1) \Float4;

ClearRawBytes raw_data \FromIndex := 5;
```

In die ersten 4 Byte wird der Wert von `integer` übernommen (beginnend mit Index 1) und in die nächsten 4 Byte (beginnend mit Index 5) der Wert von `float`.

Die letzte Instruktion im Beispiel löscht die Inhalte von `raw_data`, beginnend mit Index 5, d. h. `float` wird gelöscht, aber `integer` wird in `raw_data` beibehalten.

Die aktuelle Länge gültiger Byte in`raw_data` wird auf 4 gesetzt.

Argumente

```
ClearRawBytes RawData [ \FromIndex ]
```

`RawData`

Datentyp: `rawbytes`

`RawData` ist der Daten-Container, der gelöscht wird.

`[ \FromIndex ]`

Datentyp: `num`

Mit `\FromIndex` wird angegeben, ab welchem Index mit dem Löschen des Inhalts von `RawData` begonnen werden soll. Es wird alles bis zum Ende gelöscht.

Wenn für `\FromIndex` kein Wert angegeben wird, werden alle Daten beginnend mit Index 1 gelöscht.

Programmabarbeitung

Die Daten ab Index 1 (Standard) oder ab `\FromIndex` in der angegebenen Variablen werden auf 0 zurückgesetzt.

Die aktuelle Länge gültiger Bytes in der angegebenen Variablen wird auf 0 (Standard) oder auf (`FromIndex`- 1) gesetzt, wenn `\FromIndex` programmiert wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
ClearRawBytes
  [RawData ':=' ] < variable (VAR) of rawbytes>
  ['\'FromIndex ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.47 ClkReset - Stellt eine Uhr zurück, die zur Zeitnahme verwendet wird

Verwendung

`ClkReset` wird verwendet, um eine Uhr zurückzustellen, die als Stoppuhr für eine Zeitnahme dient.

Diese Instruktion kann vor dem Start einer Uhr gewährleisten, dass die Uhr auf 0 eingestellt ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ClkReset`:

Beispiel 1:

```
ClkReset clock1;
```

Die Uhr `clock1` wird zurückgestellt.

Argumente

```
ClkReset Clock
```

`Clock`

Datentyp: `clock`

Der Name der zurückzustellenden Uhr.

Programmabarbeitung

Eine zurückgestellte Uhr wird auf 0 eingestellt.

Eine laufende Uhr wird gestoppt und dann zurückgestellt.

Syntax

```
ClkReset
  [ Clock ':=' ] < variable (VAR) of clock > ';'
```

Weitere Informationen

Information über	Siehe
Weitere Instruktionen für Uhren	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.48 ClkStart - Starten einer Uhr, die zur Zeitnahme verwendet wird

Verwendung

`ClkStart` wird verwendet, um eine Uhr zu starten, die als Stoppuhr für eine Zeitnahme dient.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ClkStart`:

Beispiel 1:

```
ClkStart clock1;
```

Die Uhr `clock1` wird gestartet.

Argumente

```
ClkStart Clock
```

`Clock`

Datentyp: `clock`

Der Name der zu startenden Uhr.

Programmabarbeitung

Wenn eine Uhr gestartet wird, läuft sie und zählt die Sekunden, bis sie gestoppt wird.

Eine Uhr läuft auch nach dem Stopp des Programms weiter, von dem sie gestartet wurde. Jedoch ist eventuell das Ereignis nicht mehr gültig, dessen zeitlichen Ablauf Sie steuern wollten. Wenn das Programm z. B. die Wartezeit für eine Eingabe misst, kann die Eingabe erfolgt sein, während das Programm gestoppt war. Das Programm kann dieses eingetretene Ereignis nicht "sehen", während es gestoppt war.

Eine Uhr läuft weiter, auch wenn das Robotersystem ausgeschaltet wird, sofern die Batterie den Programmspeicher puffert, der die clock-Variable enthält.

Eine laufende Uhr kann abgelesen, gestoppt oder zurückgestellt werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_OVERFLOW`	Die Uhr läuft 4.294.967 Sekunden lang (49 Tage, 17 Stunden, 2 Minuten und 47 Sekunden), dann entsteht ein Überlauf.

Weitere Beispiele

Weitere Beispiele für die Instruktion `ClkStart` sind unten aufgeführt.

Beispiel 1:

```
VAR clock clock2;
VAR num time;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.48 ClkStart - Starten einer Uhr, die zur Zeitnahme verwendet wird

RobotWare - OS

Fortsetzung

```
ClkReset clock2;
ClkStart clock2;
WaitUntil di1 = 1;
ClkStop clock2;
time:=ClkRead(clock2);
```

Die Zeitdauer, bis `di1` 1 ist, wird gemessen.

Syntax

```
ClkStart
  [ Clock ':=' ] < variable (VAR) of clock >';'
```

Weitere Informationen

Information über	Siehe
Weitere Instruktionen für Uhren	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.49 ClkStop - Stoppen einer Uhr, die zur Zeitnahme verwendet wird

Verwendung

`ClkStop` wird verwendet, um eine Uhr zu stoppen, die als Stoppuhr für eine Zeitnahme dient.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ClkStop`:

```
ClkStop clock1;
```

Die Uhr `clock1` wird gestoppt.

Argumente

```
ClkStop Clock
```

`Clock`

Datentyp: `clock`

Der Name der zu stoppenden Uhr.

Programmabarbeitung

Eine gestoppte Uhr läuft nicht weiter.

Sie kann dann abgelesen, erneut gestartet oder zurückgestellt werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_OVERFLOW`	Die Uhr läuft 4.294.967 Sekunden lang (49 Tage, 17 Stunden, 2 Minuten und 47 Sekunden), dann entsteht ein Überlauf.

Syntax

```
ClkStop
[ Clock ':=' ] < variable (VAR) of clock >';'
```

Weitere Informationen

Information über	Siehe
Weitere Instruktionen für Uhren	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Beispiele	*ClkStart - Starten einer Uhr, die zur Zeitnahme verwendet wird auf Seite 149*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.50 Schließen - Schließt eine Datei oder ein E/A-Gerät

Verwendung

`Close` wird verwendet, um eine Datei oder ein E/A-Gerät zu schließen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Close`:

Beispiel 1:

```
Close channel2;
```

Das E/A-Gerät auf das sich `channel2` bezieht, ist geschlossen.

Argumente

```
Close IODevice
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) der Datei oder des E/A-Geräts, die/der geschlossen werden soll.

Programmabarbeitung

Die angegebene Datei bzw. der angegebene E/A-Gerät wird geschlossen und muss vor einem Lese- oder Schreibvorgang wieder geöffnet werden. Wenn die Datei/der Kanal bereits geschlossen ist, wird die Instruktion ignoriert.

Syntax

```
Close
  [IODevice ':='] <variable (VAR) of iodev>';'
```

Weitere Informationen

Information über	Siehe
Eine Datei oder ein E/A-Gerät öffnen	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.51 CloseDir - Schließen eines Verzeichnisses

Verwendung

`CloseDir` wird verwendet, um ein Verzeichnis zu schließen, das mit `OpenDir` geöffnet wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CloseDir`:

Beispiel 1:

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
CloseDir directory;
ENDPROC
```

Dieses Beispiel gibt die Namen aller Dateien bzw. Unterverzeichnisse unter dem angegebenen Verzeichnis aus.

Argumente

```
CloseDir Dev
```

`Dev`

Datentyp: `dir`

Eine Variable mit Verweis auf das Verzeichnis, das mit der Instruktion `OpenDir` abgerufen wurde.

Syntax

```
CloseDir
  [ Dev ':=' ] < variable (VAR) of dir>';'
```

Weitere Informationen

Information über	Siehe
Verzeichnis	*dir – Dateiverzeichnisstruktur auf Seite 1689*
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Öffnen eines Verzeichnisses	*OpenDir - Öffnen eines Verzeichnisses auf Seite 497*
Lesen eines Verzeichnisses	*ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis auf Seite 1459*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.52 Comment - Kommentar

Verwendung

`Comment` wird nur verwendet, um die Verständlichkeit eines Programms zu verbessern. Dies hat keinen Einfluss auf die Abarbeitung des Programms.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Comment`:

Beispiel 1:

```
! Goto the position above pallet
MoveL p100, v500, z20, tool1;
```

Ein Kommentar wird in das Programm eingefügt, damit es leichter verständlich ist.

Argumente

```
! Comment
```

`Comment`

Zeichenfolge

Beliebiger Text.

Programmabarbeitung

Das Abarbeiten dieser Instruktion hat keinerlei Auswirkungen.

Einschränkungen

Kommentare in einem Datensatz

In einer Datensatzdefinition ist kein Kommentar erlaubt, der sich in einer separaten Zeile befindet, es sei denn, er befindet sich in der letzten Zeile.

```
RECORD my_rec
  ! DISALLOWED COMMENT
  num mynum; ! allowed comment (not separate line)
  string mystring;
  ! allowed comment on last line
ENDRECORD
```

Syntax

```
'!' {<character>} <newline>
```

Weitere Informationen

Information über	Siehe
In einem Kommentar zulässige Zeichen	*Technisches Referenzhandbuch - RAPID Overview*
Kommentare in Daten und Routinen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.53 Compact IF - Wenn eine Bedingung erfüllt ist, dann... (eine Instruktion)

Verwendung

Mit `Compact IF` wird eine einzelne Instruktion nur dann abgearbeitet, wenn eine vorgegebene Bedingung erfüllt ist.

Wenn verschiedene Instruktionen abhängig davon abzuarbeiten sind, ob die festgelegte Bedingung erfüllt ist oder nicht, wird die `IF`-Instruktion verwendet.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `CompactIF`:

Beispiel 1:

```
IF reg1 > 5 GOTO next;
```

Wenn `reg1` größer als `5` ist, wird die Programmabarbeitung mit dem `next`-Label fortgesetzt.

Beispiel 2:

```
IF counter > 10 Set do1;
```

Das Signal `do1` wird gesetzt, wenn `counter > 10`.

Argumente

```
IF Condition ...
```

Condition

Datentyp: `bool`

Die Bedingung muss erfüllt sein, bevor die Instruktion abgearbeitet wird.

Syntax

```
IF <conditional expression> ( <instruction> | <SMT>) ';'
```

Weitere Informationen

Information über	Siehe
Bedingungen (logische Ausdrücke)	*Technisches Referenzhandbuch - RAPID Overview*
IF mit mehreren Instruktionen	*IF - Wenn eine Bedingung erfüllt ist, dann...; sonst... auf Seite 258*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.54 ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung

Verwendung

`ConfJ` (*Configuration Joint*) legt fest, ob die Konfiguration des Roboters während der achsenweisen Bewegung gesteuert werden soll. Falls sie nicht gesteuert wird, kann der Roboter in manchen Fällen eine andere als die programmierte Konfiguration verwenden.

Mit `ConfJ \Off` kann das Robotersystem die Hauptachsenkonfiguration nicht wechseln, sondern sucht nach einer Lösung mit der aktuellen Hauptachsenkonfiguration. Daher wird die Handgelenkkonfiguration mit der größten Übereinstimmung für die Achsen 4 und 6 verwendet.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Hinweis

Für IRB5400 und IRB5500 ist die Roboterüberwachung stets abgeschaltet, unabhängig von der Angabe in `ConfJ`.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ConfJ`:

Beispiel 1:

```
ConfJ \Off;
MoveJ *, v1000, fine, tool1;
```

Der Roboter bewegt sich in die programmierte Position und Orientierung. Wenn sich diese Position auf verschiedene Arten mit unterschiedlichen Achsenkonfigurationen erreichen lässt, wird die nächstgelegene Position gewählt.

Beispiel 2:

```
ConfJ \On;
MoveJ *, v1000, fine, tool1;
```

Der Roboter bewegt sich zur programmierten Position, Orientierung und Achskonfiguration.

Argumente

```
ConfJ [\On] | [\Off]
```

`[\On]`

Datentyp: `switch`

Der Roboter bewegt sich zu der programmierten Position mit den Konfigurationsparametern, die gleich oder ähnlich den vorgegebenen Parametern in den `confdata` sind.

Wenn eine Programmverschiebung oder Pfadkorrektur aktiv ist, ist das Risiko für große Bewegungen höher, weil die Konfigurationsdaten des Programms auf der ursprünglichen Position basieren.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der IRB 5400-Roboter bewegt sich in die programmierte Achsenkonfiguration oder eine Achsenkonfiguration, die nahe der programmierten liegt.

`[\Off]`

Datentyp: `switch`

Der Roboter bewegt sich zu der programmierten Position unter Verwendung der nächstgelegenen Achskonfiguration.

Programmabarbeitung

Die Konfiguration gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue `ConfJ`-Instruktion abgearbeitet wird.

Wenn das Argument `\On` (oder kein Argument) ausgewählt wird, bewegt sich der Roboter zu der programmierten Position mit den Konfigurationsparametern, die gleich oder ähnlich wie die vorgegebenen Konfigurationsparameter sind.

Wenn eine Programmverschiebung oder Pfadkorrektur aktiv ist, ist das Risiko für große Bewegungen höher, weil die Konfigurationsdaten des Programms auf der ursprünglichen Position basieren.

Wenn das Argument `\Off` gewählt wird, bewegt sich der Roboter stets in die nächstgelegene Achsenkonfiguration. Diese kann sich von der programmierten Konfiguration unterscheiden, wenn die Konfiguration manuell falsch eingegeben oder eine Programmverschiebung ausgeführt wurde.

Die Steuerung der Konfiguration (`ConfJ \On`) ist standardmäßig aktiv. Dies wird automatisch eingestellt.

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Syntax

```
ConfJ
  [ '\' On] | [ '\' Off]';'
```

Weitere Informationen

Information über	Siehe
Handhabung verschiedener Konfigurationen	*Technisches Referenzhandbuch - RAPID Overview*
Roboterkonfiguration bei linearer Bewegung	*ConfL - Überwachen der Konfiguration während linearer Bewegung auf Seite 158*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Konfigurationsdaten des Roboters	*confdata – Konfigurationsdaten des Roboters auf Seite 1677*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.55 ConfL - Überwachen der Konfiguration während linearer Bewegung

Verwendung

`ConfL` *Configuration Linear*) legt fest, ob die Konfiguration des Roboters während der linearen oder kreisförmigen Bewegung überwacht werden soll. Ohne Überwachung kann sich die Konfiguration bei der Abarbeitung von der programmierten Konfiguration unterscheiden. Es können auch unerwartete Roboterbewegungen auftreten, wenn der Modus in achsenweise Bewegung geändert wird.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Hinweis

Für IRB5400 und IRB5500 ist die Roboterüberwachung stets abgeschaltet, unabhängig von der Angabe in `ConfL`.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ConfL`:

Beispiel 1:

```
ConfL \On;
MoveL *, v1000, fine, tool1;
```

Die Programmabarbeitung stoppt, sobald die programmierte Konfiguration nicht von der aktuellen Position aus erreicht werden kann.

Beispiel 2:

```
SingArea \Wrist;
ConfL \On;
MoveL *, v1000, fine, tool1;
```

Der Roboter bewegt sich in die programmierte Position, Orientierung und Handgelenkachsenkonfiguration. Falls dies nicht möglich ist, stoppt die Programmabarbeitung.

Beispiel 3

```
ConfL \Off;
MoveL *, v1000, fine, tool1;
```

Der Roboter bewegt sich in die programmierte Position und Orientierung, aber an die nächstmögliche Achsenkonfiguration, die sich von der programmierten Konfiguration unterscheiden kann.

Argumente

```
ConfL [\On]|[\Off]
```

`[ \On ]`

Datentyp: `switch`

Die Roboterkonfiguration wird überwacht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \Off ]`

Datentyp: `switch`

Die Roboterkonfiguration wird nicht überwacht.

Programmabarbeitung

Die Konfiguration gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue `ConfL`-Instruktion abgearbeitet wird.

Bei einer linearen Bewegung bewegt sich der Roboter stets in die programmierte Position und Orientierung mit der möglichst genau entsprechenden Achsenkonfiguration. Wenn das Argument `\On` (oder kein Argument) verwendet wird, stoppt die Programmabarbeitung, sobald das Risiko besteht, dass die Konfiguration der programmierten Position von der aktuellen Position aus nicht erreicht wird. Die Weise, auf die darüber entschieden wird, variiert je nach Robotertyp, siehe *confdata – Konfigurationsdaten des Roboters auf Seite 1677*.

Bevor eine angeforderte Bewegung begonnen wird, wird eine Überprüfung vorgenommen, um festzustellen, ob es möglich ist, die programmierte Konfiguration zu erreichen. Wenn dies nicht möglich ist, wird das Programm angehalten. Wenn die Bewegung abgeschlossen ist (in einer Zone oder Feinpunkt), wird auch überprüft, ob der Roboter die programmierte Konfiguration erreicht hat.

Wenn `SingArea \Wrist` verwendet wird, bewegt sich der Roboter immer in die programmierte Handgelenkachsenkonfiguration.

Wenn das Argument `\Off` verwendet wird, gibt es keine Überwachung.

Nach einem Stopp, der durch einen Konfigurationsfehler hervorgerufen wurde, ist es möglich, das RAPID-Programm im manuellen Modus neu zu starten. Bedenken Sie, dass sich der Roboter in diesem Fall aufgrund des gemeldeten Fehlers nicht in die richtige Konfiguration bewegen wird.

Wenn `ConfL \Off` mit einer großen Bewegung verwendet wird, kann sie unmittelbar oder später Stopps im Programm mit dem Fehler `50050 Position outside reach` oder `50080 Position not compatible` verursachen.

Die Überwachung der Konfiguration (`ConfL \On`) ist standardmäßig aktiv. Dies wird automatisch eingestellt.

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.



Tipp

Eine einfache Daumenregel, um Probleme sowohl bei `ConfL\On` als auch `\Off` zu vermeiden: fügen Sie dazwischen liegende Punkte ein, um die Bewegung jeder Achse auf weniger als 180 Grad zwischen Punkten einzustellen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.55 ConfL - Überwachen der Konfiguration während linearer Bewegung

RobotWare - OS

Fortsetzung



Tipp

In einem Programm mit `ConfL \Off` empfiehlt es sich, vor Bewegungen in verschiedenen Programmteilen Startpunkte mit bekannten Konfigurationspunkten mit "`ConfJ \On` und `MoveJ`" oder "`ConfL \On` und `SingArea \Wrist` und `MoveL`" zu haben.

Syntax

```
ConfL
  [ '\' On] | [ '\' Off]';'
```

Weitere Informationen

Information über	Siehe
Handhabung verschiedener Konfigurationen	*Technisches Referenzhandbuch - RAPID Overview*
Roboterkonfiguration bei Achsenbewegung	*ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung auf Seite 156*
Definieren der Interpolation in der Nähe singulärer Punkte	*SingArea - Definieren der Interpolation in der Nähe singulärer Punkte auf Seite 736*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Konfigurationsdaten des Roboters	*confdata – Konfigurationsdaten des Roboters auf Seite 1677*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.56 CONNECT - Verbinden eines Interrupts mit einer Interrupt-Routine

Verwendung

`CONNECT` wird verwendet, um die Identität eines Interrupts zu ermitteln und ihn mit einer Interrupt-Routine zu verbinden.

Der Interrupt wird durch Anfordern eines Interrupt-Ereignisses und Angeben seiner Identität definiert. Wenn dann dieses Ereignis eintritt, wird die Interrupt-Routine automatisch abgearbeitet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CONNECT`:

Beispiel 1:

```
VAR intnum feeder_low;
PROC main()
  CONNECT feeder_low WITH feeder_empty;
  ISignalDI di1, 1 , feeder_low;
  ...
```

Die Interrupt-Identität `feeder_low` wird erstellt und mit der Interrupt-Routine `feeder_empty` verbunden. Ein Interrupt tritt auf, wenn Eingang `di1` auf 1 geht. Das heißt, wenn dieses Signal auf 1 geht, wird die Interrupt-Routine `feeder_empty` abgearbeitet.

Argumente

```
CONNECT Interrupt WITH Trap routine
```

`Interrupt`

Datentyp: `intnum`

Die Variable, der die Identität des Interrupts zugewiesen werden soll. Diese darf nicht in einer Routine (Routinendaten) deklariert werden.

`Trap routine`

Identifier

Der Name der Interrupt-Routine.

Programmabarbeitung

Der Variablen wird eine Interrupt-Identität zugewiesen, die dann beim Anfordern oder Deaktivieren von Interrupts verwendet werden soll. Diese Identität ist auch mit der angegebenen Interrupt-Routine verbunden.

Hinweis

Alle Interrupts in einem Task werden abgebrochen, wenn der Programmzeiger für diesen Task auf "main" gesetzt wird, und müssen neu verbunden werden. Ein Stromausfall oder ein **Neustart** hat keinerlei Auswirkungen auf die Interrupts.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.56 CONNECT - Verbinden eines Interrupts mit einer Interrupt-Routine
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_ALRDYCNT	Die Interrupt-Variable ist bereits mit einer TRAP-Routine verbunden.
ERR_CNTNOTVAR	Die Interrupt-Variable stellt keinen Variablenverweis dar.
ERR_INOMAX	Es sind keine Interruptnummern mehr verfügbar.

Einschränkungen

Ein Interrupt (Interrupt-Identität) kann nicht mit mehreren Interrupt-Routinen verbunden sein. Verschiedene Interrupts können jedoch mit derselben Interrupt-Routine verbunden werden.

Wenn ein Interrupt mit einer Interrupt-Routine verbunden wurde, kann er nicht mit einer anderen Routine verbunden oder an diese übertragen werden. Er muss zuvor mit der Instruktion IDelete gelöscht werden.

Interrupts, die nach dem Stoppen der Programmabarbeitung erfolgen oder bis dahin nicht behandelt wurden, werden ignoriert. Die Interrupts werden beim Anhalten des Programms nicht beachtet. Interrupts, die als sicher eingestellt wurden, werden beim Programmstopp nicht ignoriert. Sie werden bearbeitet, wenn das Programm wieder gestartet wird.

Syntax

```
CONNECT <connect target> WITH <trap>';'
<connect target> ::= <variable>
                   | <parameter>
                   | <VAR>
<trap> ::= <identifier>
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Datentyp für Interrupt	*intnum - Interrupt-Identität auf Seite 1716*
Abbrechen eines Interrupts	*IDelete - Abbrechen eines Interrupts auf Seite 252*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.57 ContactL – Lineare Kontaktbewegung

Verwendung

`ContactL` *(Contact Linear)* wird für den YuMi Roboter verwendet, um Kontakt mit einem Objekt an einer gewünschten Position zu erhalten und gleichzeitig den Werkzeugarbeitspunkt (TCP) linear zu bewegen.

Die Kollisionserkennungsstufe wird auf ihren Maximalwert angehoben und während der Bewegung überwacht der Roboter das interne Drehmoment und vergleicht ihn mit einem Niveau des Drehmoments, das vom Anwender vorgegeben wird. Wenn das angeforderte Drehmomentniveau des Anwenders erreicht wird, führt der Roboter einen abrupten Stopp durch und fährt mit dem übrigen Programm fort.

Diese Instruktion eignet sich typischerweise für den Einsatz eines Werkzeugs, wenn der Roboter ein Objekt einsetzen muss.

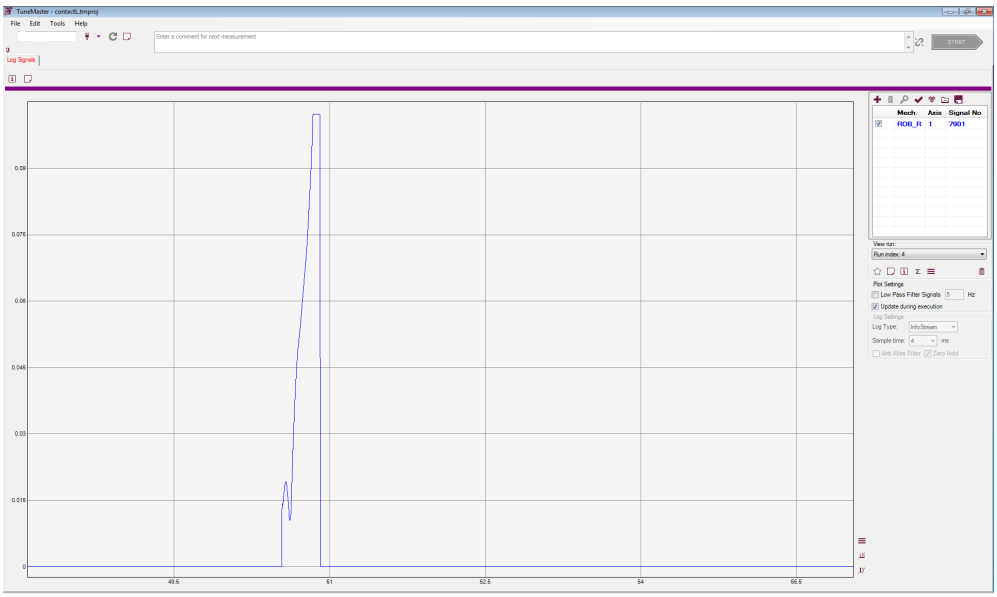
Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in der Bewegungstask in einem *MultiMove*-System verwendet werden.

Die Maximalgeschwindigkeit für eine `ContactL` Instruktion 1000 beträgt mm/s.

> **Hinweis**
>
> Während der Programmierung wird für den ersten Test eine langsame Geschwindigkeit empfohlen, <100 mm/s, die dann schrittweise auf den gewünschten Wert erhöht wird.

Beschreibung

Um den Wert des Drehmomentniveaus `desiredTorque` herauszufinden, ist es hilfreich, die Anwendung zu prüfen und ein internes Testsignal, Signal 7901, mithilfe von TuneMaster anzuzeigen.



xx1500000649

Weitere Informationen zu TuneMaster finden sie im Hilfebereich der Anwendung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.57 ContactL – Lineare Kontaktbewegung

YuMi

Fortsetzung

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ContactL`:

Siehe auch *Weitere Beispiele auf Seite 167*.

Beispiel 1:

```
desiredTorque := 0.1;
ContactL \DesiredTorque:=desiredTorque, p10, v100, tool1;
```

Das TCP von `tool1` wird mit der Geschwindigkeit `p10` linear an die Position `v100` bewegt. Wenn der Wert des internen Drehmomentniveaus `desiredTorque` das vom Anwender festgelegte Level überschreitet, führt der Roboter einen abrupten Stopp durch und das Programm wird von der Position, in der der Roboter gestoppt wurde, fortgesetzt.

Das Argument `DesiredTorque` ist optional. Wenn `DesiredTorque`weggelassen wird, hebt die Anweisung `ContactL` nur das Kollisionserkennungslevel auf den Maximalwert an, z.B. die Möglichkeit der Druckbeibehaltung eines Objekts während der Bewegung des TCP.

Wenn der `desiredTorque` nicht erreicht wird, wenn der Roboter die gewünschte Position einnimmt, wird ein Ausführungsfehler gemeldet und das System wird mit einem Ereignisprotokoll unterbrochen. Deshalb wird empfohlen, in solchen Fällen eine Fehlerbehandlung zu implementieren, siehe *Fehlerbehandlung auf Seite 166*.

Beispiel 2:

```
ContactL RelTool (CRobT(),5,5,0), v100, \Zone:=z10, tool1;
```

Der Roboter wird in eine Position bewegt, die sich 5 mm von seiner aktuellen Position in x-Richtung und 5 mm von seiner aktuellen Position in y-Richtung des Werkzeugs befindet. Wenn das `Zone` Argument weggelassen wird, verwendet die `ContactL`Instruktion standardmäßig einen Feinpunkt.

In dem Beispiel wird das Argument `DesiredTorque` weggelassen. Die Instruktion hebt nur das Kollisionserkennungslevel auf den Maximalwert an und die `ContactL`-Instruktion funktioniert ähnlich wie eine `MoveL`-Instruktion.

Beispiel 3

```
desiredTorque := 0.9;
ContactL \DesiredTorque:=desiredTorque, p10, v100, tool1;
ContactL RelTool (CRobT(),5,5,0), v100, \Zone:=z10, tool1;
ContactL RelTool (CRobT(),5,5,-10), v100, \Zone:=z10, tool1;
MoveL ...
```

Es ist wichtig zu beachten, dass die `ContactL`-Instruktion sowohl während des Kontakts als auch beim Verlassen des Kontakts angewandt wird. Eine normale Bewegungsinstruktion löst höchstwahrscheinlich die Bewegungsüberwachung aus.

Argumente

```
ContactL [\DesiredTorque] ToPoint [\ID] Speed [\Zone] Tool [\WObj]
```

`[ \DesiredTorque ]`

Datentyp: `num`

Benutzerdefiniertes gewünschtes Niveau des Drehmoments

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ContactL verwendet immer Feinpunkte als Zonendaten für den Zielpunkt, wenn DesiredTorque definiert ist. Wenn DesiredTorque weggelassen wird, hebt die ContactL-Instruktion nur die Stufe der Kollisionserkennung und überprüft nicht das interne Drehmomentniveau.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \Zone ]`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn und werden nur verwendet, wenn DesiredTorque weggelassen wird.

Wenn das `[\Zone]` -Argument weggelassen wird, verwendet die ContactL-Anweisung standardmäßig einen Feinpunkt.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.57 ContactL – Lineare Kontaktbewegung
YuMi
Fortsetzung

Programmabarbeitung

Siehe die Instruktion `MoveL` für weitere Informationen über lineare Bewegung.

Die Roboterbewegung wird unterbrochen, wenn das interne Niveau des Drehmoments das benutzerdefinierte Niveau des Drehmoments überschritten hat, vorausgesetzt, dass Argument `DesiredTorque` definiert wurde. Andernfalls fährt die Roboterbewegung stets zum programmierten Zielpunkt fort.

Wenn das Argument `DesiredTorque` weggelassen wird, wird das Kollisionserkennungslevel auf den Maximalwert angehoben und es wird keine Überwachung des internen Drehmomentniveaus durchgeführt, d.h. die Druckbeibehaltung eines Objekts während der Bewegung des TCP wird ermöglicht.

Fehlerbehandlung

Während einer `ContactL` wird ein Fehler gemeldet, wenn:

- `ContactL` erreicht den im Argument `ToPoint`festgelegten Punkt, ohne dass der vom Benutzer festgelegte `DesiredTorque` erreicht wird. Dadurch wird der Fehler `ERR_CONTACTL` generiert.

Fehler können abhängig vom gewählten Abarbeitungsmodus auf verschiedene Weise behandelt werden:

- **Kontinuierlich vorwärts/Instruktion vorwärts**:

  Es wird keine Position zurückgegeben und die Bewegung fährt stets zum programmierten Zielpunkt fort. Die Systemvariable `ERRNO` wird auf `ERR_CONTACTL` gesetzt und der Fehler kann in der Fehlerbehandlung der Routine behandelt werden.
- **Instruktions rückwärts**:

  Bei der Rückwärtsabarbeitung führt die Instruktion die Bewegung ohne jegliche Drehmomentüberwachung aus.

Beispiel

```
VAR num desiredTorque;
...
desiredTorque := 0.1;
MoveL p10, v100, fine, tool1;
ContactL \DesiredTorque:=desiredTorque, p20, v100, tool1;
...
ERROR
  IF ERRNO=ERR_CONTACTL THEN
    StorePath;
    MoveL p10, v100, fine, tool1;
    RestoPath;
    ClearPath;
    StartMove;
    RETRY;
  ELSE
    Stop;
  ENDIF
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Roboter bewegt sich von Position `p10` zu `p20`. Wenn der Roboter `p20` erreicht, ohne den vom Benutzer festgelegten `DesiredTorque` zu erreichen, bewegt sich der Roboter zurück zu `p10` und versucht es erneut.

Weitere Beispiele

Weitere Beispiele für die Instruktion `ContactL` sind unten aufgeführt.

Beispiel 1:

```
ContactL p10, v100, \Zone:=z10, tool1;
```

Der TCP `tool1` wird mit der Geschwindigkeit `p10`und einer Zonengröße von 10 mm linear an die Position `v100` bewegt.

Da das Argument `DesiredTorque` weggelassen wird, hebt die Anweisung `ContactL` nur das Kollisionserkennungslevel auf den Maximalwert an, und überprüft nicht das interne Drehmomentniveau.

Einschränkungen

Die Instruktionen `ContactL` können nur für YuMi-Roboter verwendet werden.

Syntax

```
ContactL
  ['\' DesiredTorque ',']
  [ToPoint ':='] <expression (IN) of robtarget>
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' Zone ':=' <expression (IN) of zonedata>]','
  [Tool ':=' ] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]';'
```

Weitere Informationen

Information über	Siehe
Verwendung der IRB 14000	*Bedienungsanleitung - IRB 14000*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lineare Bewegung des Roboters	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
Verwenden von Fehlerbehandlungsroutinen	*Technisches Referenzhandbuch - RAPID Overview*
Lineare Bewegung	
Bewegung im Allgemeinen	
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.57 ContactL – Lineare Kontaktbewegung

YuMi

Fortsetzung

Information über	Siehe
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.58 CopyFile - Kopieren einer Datei

Verwendung

`CopyFile` wird verwendet, um von einer bereits vorhandenen Datei eine Kopie zu erstellen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CopyFile`:

Beispiel 1:

CopyFile "HOME:/myfile", "HOME:/yourfile";

Von der Datei `myfile` wird eine Kopie mit dem Namen `yourfile` angefertigt. Beide Dateien sind vom Inhalt her identisch.

```
CopyFile "HOME:/myfile", "HOME:/mydir/yourfile";
```

Die Datei `myfile` wird unter dem Namen `yourfile` in das Verzeichnis `mydir` kopiert.

Argumente

```
CopyFile OldPath NewPath
```

`OldPath`

Datentyp: `string`

Der vollständige Pfad der zu kopierenden Datei.

`NewPath`

Datentyp: `string`

Der vollständige Pfad, unter dem die Kopie der Datei gespeichert werden soll.

Programmabarbeitung

Von der unter `OldPath` angegebenen Datei wird unter `NewPath` eine Kopie gespeichert.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_FILEACC`	Die in `OldPath` angegebene Datei existiert nicht.
`ERR_FILEEXIST`	Die unter `NewPath` angegebene Datei ist bereits vorhanden.

Syntax

```
CopyFile
[ OldPath ':=' ] < expression (IN) of string > ','
[ NewPath ':=' ] < expression (IN) of string >';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.58 CopyFile - Kopieren einer Datei

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.59 CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten

Verwendung

`CopyRawBytes` wird verwendet, um den vollständigen Inhalt oder Teile des Inhalts von einer `rawbytes`-Variablen in eine andere zu kopieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CopyRawBytes`:

Beispiel 1:

```
VAR rawbytes from_raw_data;
VAR rawbytes to_raw_data;
VAR num integer := 8
VAR num float := 13.4;

ClearRawBytes from_raw_data;
PackRawBytes integer, from_raw_data, 1 \IntX := DINT;
PackRawBytes float, from_raw_data, (RawBytesLen(from_raw_data)+1)
     \Float4;
CopyRawBytes from_raw_data, 1, to_raw_data, 3,
     RawBytesLen(from_raw_data);
```

In diesem Beispiel wird die Variable `from_raw_data` des Typs `rawbytes` zunächst gelöscht, also alle Bytes auf 0 gesetzt. Dann wird in die ersten 4 Bytes der `integer`-Wert übernommen und die nächsten 4 Bytes erhalten den `float`-Wert.

Nachdem `from_raw_data` mit Daten gefüllt ist, wird der Inhalt (8 Bytes) beginnend mit Position `3` in `to_raw_data` kopiert.

Argumente

```
CopyRawBytes FromRawData FromIndex ToRawData ToIndex[\NoOfBytes]
```

`FromRawData`

Datentyp: `rawbytes`

`FromRawData` ist der Daten-Container, aus dem die `rawbytes`-Daten kopiert werden sollen.

`FromIndex`

Datentyp: `num`

`FromIndex` ist diejenige Position in `FromRawData`, an der die zu kopierenden Daten beginnen. Die Indexierung beginnt bei 1.

`ToRawData`

Datentyp: `rawbytes`

`ToRawData` ist der Daten-Container, in den die `rawbytes`-Daten kopiert werden sollen.

`ToIndex`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.59 CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten

RobotWare - OS

Fortsetzung

`ToIndex` ist diejenige Position in `ToRawData`, an der die zu kopierenden Daten platziert werden. Es wird alles bis zum Ende kopiert. Die Indexierung beginnt bei 1.

`[\NoOfBytes]`

Datentyp: `num`

Der unter `\NoOfBytes` angegebene Wert entspricht der Anzahl an Bytes, die von `FromRawData` nach `ToRawData` kopiert werden sollen.

Wenn für `\NoOfBytes` kein Wert angegeben wird, werden alle Bytes ab `FromIndex` bis zum Ende der aktuellen Länge gültiger Bytes in `FromRawData` kopiert.

Programmabarbeitung

Bei der Programmausführung werden die Daten aus einer `rawbytes`-Variablen in eine andere kopiert.

Die aktuelle Länge gültiger Bytes in der Variablen `ToRawData` wird gesetzt auf:

- (`ToIndex` + copied_number_of_bytes - 1)
- Die aktuelle Länge gültiger Bytes in der Variablen `ToRawData` wird nicht geändert, wenn der komplette Kopiervorgang innerhalb der alten aktuellen Länge gültiger Bytes in der Variablen `ToRawData` durchgeführt wird.

Einschränkungen

`CopyRawBytes` kann nicht verwendet werden, um Daten von einer `rawbytes`-Variablen an eine andere Stelle derselben `rawbytes`-Variablen zu kopieren.

Syntax

```
CopyRawBytes
  [FromRawData ':=' ] < variable (VAR) of rawbytes> ','
  [FromIndex ':=' ] < expression (IN) of num> ','
  [ToRawData ':=' ] < variable (VAR) of rawbytes> ','
  [ToIndex ':=' ] < expression (IN) of num>
  ['\'NoOfBytes ':=' < expression (IN) of num> ]';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.60 CornerPathWarning – Zonenbahnwarnungen anzeigen oder verbergen

Verwendung

`CornerPathWarning` wird verwendet, um Zonenbahnwarnungen (Ereignisprotokoll 50024) für alle nachfolgenden Bewegungsinstruktionen zu aktivieren/deaktivieren.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `CornerPathWarning`.

Beispiel 1:

```
CornerPathWarning TRUE;
```

Aktivieren von Zonenbahnwarnungen

Beispiel 2:

```
PROC main()
  ! Deactivate corner path warning on all
  ! subsequent movement instructions
  CornerPathWarning FALSE;
  ...
  ! Check if warning is suppressed
  IF C_MOTSET.corner_path_warn_suppress=TRUE THEN
    CornerPathWarning TRUE;
  ENDIF
  MyProcess;
```

Deaktivieren Sie Zonenbahnwarnungen beim Start des Programms. Prüfen Sie später, ob die Zonenbahnwarnung unterdrückt wird. Wird diese unterdrückt, aktivieren Sie die Zonenbahnwarnung, bevor Sie `MyProcess` aufrufen.

Argumente

```
CornerPathWarning Active
```

`Active`

Datentyp: `bool`

Legt fest, ob die Zonenbahnwarnungen aktiv sein sollen.

Programmabarbeitung

Die Einstellung gilt für die nächste auszuführende Bewegungsinstruktion, sowohl für den Roboter als die externen Achsen, bis eine neue `CornerPathWarning` ausgeführt wird.

Ein Zonenbahnfehler tritt auf, wenn der Roboter eine Verschleifzonen-Bewegungsinstruktion ausführt und die RAPID-Programmabarbeitung nicht rechtzeitig eine neue Bewegungsinstruktion liefert. Dies zwingt das System dazu, den programmierten Fly-By-Punkt in einen Feinpunkt umzuwandeln.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn Sie diese Option auf „Wahr“ setzen, wird bei jeder nachfolgenden Bewegungsinstruktion mit einem Zonenbahnfehler eine Warnung im Ereignisprotokoll angezeigt.

Wenn Sie diese Option auf „Falsch“ setzen, werden Zonenbahnfehler immer noch als Feinpunkte ausgeführt, die Warnung jedoch nicht im Ereignisprotokoll angezeigt.

Um dasselbe Verhalten wie in früheren Versionen von Robotware zu erhalten, in dem Warnungen in der Konfiguration entfernt werden konnten, wird empfohlen `CornerPathWarning FALSE;` in die Ereignisroutine zu setzen, die zu Programmstart ausgeführt wird (Ereignis `START`).

Der Standardwert (Zonenbahnfehler melden) wird automatisch eingestellt

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Hinweis

Es wird empfohlen, die Zonenbahnwarnung zu aktivieren. Schalten Sie die Zonenbahnwarnung nur in Situationen aus, in denen nur manchmal eine Warnung auftritt. Solche Situationen sind beispielsweise das Warten auf einen Eingang (von E/A, Kameras, Sensoren oder anderen externen Geräten). Eine Zonenbahnwarnung kann zu einer schlechten Prozessleistung und zu schlechteren Zykluszeiten führen. All diese Situationen müssen analysiert werden, bevor die Zonenbahnwarnung entfernt wird.

Syntax

```
CornerPathWarning
  [Active ':='] <expression (IN) of bool>';'
```

Weitere Informationen

Information über	Siehe
Bewegungsinstruktion	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Zonenbahnverhalten	Abschnitt *Interpolation of corner paths* in *Technisches Referenzhandbuch - RAPID Overview*
Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Definieren von Ereignisroutinen	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.61 CorrClear - Entfernen aller Korrekturgeneratoren

Beschreibung

`CorrClear` wird verwendet, um alle Verbindungen zu Korrekturgeneratoren zu trennen. Die Instruktion kann verwendet werden, um alle Offsets zu entfernen, die zuvor von sämtlichen Korrekturgeneratoren bereitgestellt wurden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CorrClear`:

Beispiel 1:

```
CorrClear;
```

Diese Instruktion trennt alle Verbindungen zu Korrekturgeneratoren.

Hinweis

Eine einfache Möglichkeit, um sicherzustellen, dass alle Korrekturgeneratoren (mit Korrekturen) beim Programmstart entfernt werden, ist die Ausführung von `CorrClear` in einer `START`-Ereignisroutine.

Siehe *Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Controller*.

Syntax

```
CorrClear ';'
```

Weitere Informationen

Information über	Siehe
Aufbauen einer Verbindung zu einem Korrekturgenerator	*CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*
Trennen der Verbindung zu einem Korrekturgenerator	*CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator auf Seite 182*
Schreiben an einen Korrekturgenerator	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lesen der aktuellen Gesamt-Offsets	*CorrRead - Lesen der aktuellen Gesamt-Offsets auf Seite 1252*
Korrekturdeskriptor	*corrdescr – Deskriptor des Korrekturgenerators auf Seite 1685*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.62 CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator

Verwendung

`CorrCon` wird verwendet, um eine Verbindung zu einem Korrekturgenerator aufzubauen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CorrCon`:

Siehe auch *Weitere Beispiele auf Seite 177*.

Beispiel 1

```
VAR corrdescr id;
...
CorrCon id;
```

Die Korrekturgeneratorreferenz entspricht der Variable `id`.

Argumente

```
CorrCon Descr
```

`Descr`

Datentyp: `corrdescr`

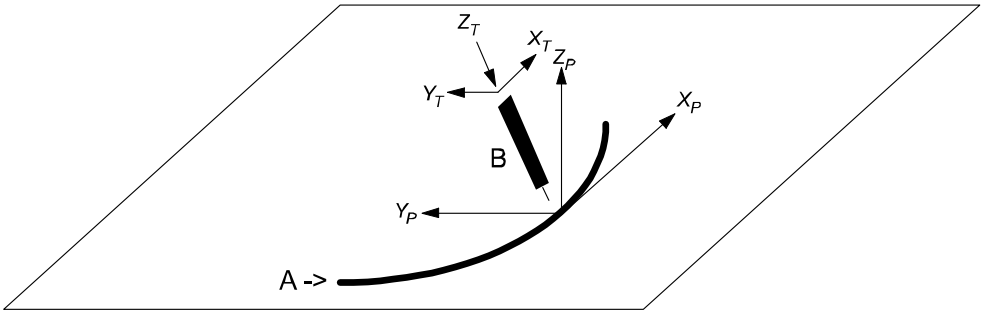
Der Deskriptor des Korrekturgenerators.

Weitere Beispiele

Weitere Beispiele für die Instruktion `CorrCon` sind unten aufgeführt.

Bahn-Koordinatensystem

Alle Pfadkorrekturen (Offsets auf der Bahn) werden im Bahn-Koordinatensystem hinzugefügt. Das Bahn-Koordinatensystem ist folgendermaßen definiert:



A	Bahnrichtung
B	Werkzeug
P	Bahn-Koordinatensystem
T	Werkzeug-Koordinatensystem

- Die Bahn-Koordinatenachse X ist als Tangente der Bahn gegeben.
- Bahnkoordinatenachse Y wird als Vektorprodukt von Bahnkoordinatenachse X und Werkzeugkoordinatenachse Z abgeleitet.

Fortsetzung auf nächster Seite

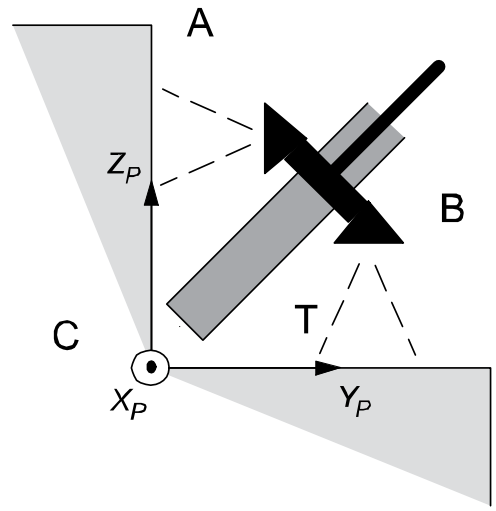
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.62 CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator
Path Offset
Fortsetzung

- Die Bahn-Koordinatenachse Z ist das Kreuzprodukt der Bahn-Koordinatenachse X und der Bahn-Koordinatenachse Y.

Beispiel für eine Anwendung

Ein Beispiel für eine Anwendung mit Pfadkorrekturen ist ein Roboter, der ein Werkzeug mit zwei daran montierten Sensoren hält, um damit die vertikalen und horizontalen Abstände zu einem Werkstück zu bestimmen. Die Abbildung unten zeigt eine Bahn-Korrekturvorrichtung.



xx0500002155

A	Sensor für horizontale Korrektur
B	Sensor für vertikale Korrektur
C	Bahn-Koordinatensystem
T	Werkzeug

Programmbeispiel

Hinweis

`hori_sig` und `vert_sig` sind Analogsignale, die in den Systemparametern definiert sind.

```
CONST num TARGET_DIST := 5;
CONST num SCALE_FACTOR := 0.5;
VAR intnum intno1;
VAR corrdescr hori_id;
VAR corrdescr vert_id;
VAR pos total_offset;
VAR pos write_offset;
VAR bool conFlag;
PROC PathRoutine()
  ! Connect to the correction generators for horizontal
  ! and vertical correction.
  CorrCon hori_id;
  CorrCon vert_id;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  conFlag := TRUE;

  ! Setup a 5 Hz timer interrupt. The trap routine will read the
  ! sensor values and compute the path corrections.
  CONNECT intno1 WITH ReadSensors;
  ITimer\Single, 0.2, intno1;

  ! Position for start of contour tracking
  MoveJ p10, v100, z10, tool1;
  ! Run MoveL with both vertical and horizontal correction.
  MoveL p20, v100, z10, tool1 \Corr;

  ! Read the total corrections added by all connected
  ! correction generators.
  total_offset := CorrRead();
  ! Write the total vertical correction on the FlexPendant.
  TPWrite "The total vertical correction is:" \Num:=total_offset.z;

  ! Disconnect the correction generator for vertical correction.
  ! Horizontal corrections will be unaffected.
  CorrDiscon vert_id;
  conFlag := FALSE;

  ! Run MoveL with only horizontal interrupt correction.
  MoveL p30, v100, fine, tool1 \Corr;
  ! Remove all outstanding connected correction generators.
  ! In this case, the only connected correction generator is
  ! the one for horizontal correction.
  CorrClear;
  ! Remove the timer interrupt.
  IDelete intno1;
ENDPROC
TRAP ReadSensors
  VAR num horiSig;
  VAR num vertSig;
  ! Compute the horizontal correction values and execute
  ! the correction.
  horiSig := hori_sig;
  write_offset.x := 0;
  write_offset.y := (hori_sig - TARGET_DIST)*SCALE_FACTOR;
  write_offset.z := 0;
  CorrWrite hori_id, write_offset;

  IF conFlag THEN
    ! Compute the vertical correction values and execute
    ! the correction.
    write_offset.x := 0;
    write_offset.y := 0;
    write_offset.z := (vert_sig - TARGET_DIST)*SCALE_FACTOR;
    CorrWrite vert_id, write_offset;
  ENDIF
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.62 CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator

Path Offset

Fortsetzung

```
    !Setup interrupt again
    IDelete intnol;
    CONNECT intno1 WITH ReadSensors;
    ITimer\single, 0.2, intno1;
  ENDTRAP
```

Programmerklärung

Zwei Korrekturgeneratoren sind mit der Instruktion `CorrCon` verbunden. Jeder dieser beiden Korrekturgeneratoren wird durch einen eindeutigen Deskriptor (`hori_id` und `vert_id`) des Typs `corrdescr` referenziert. Die beiden Sensoren verwenden jeweils einen Korrekturgenerator.

Zum Aufrufen der Interrupt-Routine `ReadSensors` mit einer Frequenz von 5 Hz wurde ein Timer-Interrupt eingerichtet. Die für die Bahnkorrektur benötigten Offsets werden in der Interrupt-Routine berechnet und mithilfe der Instruktion `CorrWrite` an den entsprechenden Korrekturgenerator geschrieben (referenziert durch die Deskriptoren `hori_id` und `vert_id`). Alle Korrekturen haben sofortige Auswirkungen auf die Bahn.

Die Instruktion `MoveL` muss zusammen mit dem switch-Argument `Corr` programmiert werden, wenn eine Bahnkorrektur verwendet werden soll. Anderenfalls werden keine Korrekturen ausgeführt.

Wenn die erste `MoveL`-Instruktion bereit ist, wird die Funktion `CorrRead` verwendet, um die Summe aller Korrekturen zu lesen (die Gesamtbahnkorrektur), die von den verbundenen Korrekturgeneratoren durchgeführt wurden. Das Ergebnis der vertikalen Gesamtbahnkorrektur wird mithilfe der Anweisung `TPWrite` an das FlexPendant übermittelt.

`CorrDiscon` trennt dann die Verbindung zum Korrekturgenerator für die vertikale Korrektur (referenziert durch den Deskriptor `vert_id`). Alle durch diesen Korrekturgenerator durchgeführten Korrekturen werden aus der Gesamtbahnkorrektur entfernt. Die Korrekturen, die durch den horizontalen Korrekturgenerator durchgeführt wurden, sind weiterhin vorhanden.

Zuletzt entfernt die Funktion `CorrClear` alle übrigen verbundenen Korrekturgeneratoren und ihre zuvor durchgeführten Korrekturen. In diesem Fall wird nur der Korrekturgenerator für horizontale Korrekturen entfernt. Der Timer-Interrupt wird mithilfe der Instruktion `IDelete` ebenfalls entfernt.

Die Korrekturgeneratoren

Die Abbildung unten zeigt die Korrekturgeneratoren.

X	Y	Z	Bahn-Koordinatenachsen
0	0	3	Vertikaler Korrekturgenerator mit der Summe aller seiner eigenen Bahnkorrekturen
0	1	0	Horizontaler Korrekturgenerator mit der Summe aller von ihm durchgeführten Bahnkorrekturen.
-	-	-	Nicht verbundener Korrekturgenerator
-	-	-	Nicht verbundener Korrekturgenerator
-	-	-	Nicht verbundener Korrekturgenerator

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

X	Y	Z	Bahn-Koordinatenachsen
0	1	3	Die Summe aller Korrekturen, die von allen verbundenen Korrekturgeneratoren durchgeführt wurden.

Einschränkungen

- Es können maximal 5 Korrekturgeneratoren gleichzeitig verbunden sein.
- Verbundene Korrekturgeneratoren bleiben bei einem Neustart der Steuerung nicht erhalten.
- Scharfe Ecken und Rückwärtsabarbeitung sollten vermieden werden, wenn ein Korrekturgenerator verwendet wird, da die Korrektur dem Koordinatensystem der Bahn hinzugefügt wird.

Syntax

```
CorrCon
  [ Descr ':=' ] < variable (VAR) of corrdescr > ';'
```

Weitere Informationen

Information über	Siehe
Trennen der Verbindung zu einem Korrekturgenerator	*CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator auf Seite 182*
Schreiben an einen Korrekturgenerator	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lesen der aktuellen Gesamt-Offsets	*CorrRead - Lesen der aktuellen Gesamt-Offsets auf Seite 1252*
Entfernen aller Korrekturgeneratoren	*CorrClear - Entfernen aller Korrekturgeneratoren auf Seite 176*
Deskriptor des Korrekturgenerators	*corrdescr – Deskriptor des Korrekturgenerators auf Seite 1685*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.63 CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator

Beschreibung

`CorrDiscon` wird verwendet, um die Verbindung zu einem früher verbundenen Korrekturgenerator zu trennen. Die Instruktion kann verwendet werden, um früher angegebene Korrekturen zu entfernen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CorrDiscon`:

Siehe auch *Weitere Beispiele auf Seite 182*.

Beispiel 1:

```
VAR corrdescr id;
...
CorrCon id;
...
CorrDiscon id;
```

`CorrDiscon` trennt die Verbindung zum früher verbundenen Korrekturgenerator, der vom Deskriptor id referenziert wird.

Argumente

```
CorrDiscon Descr
```

`Descr`

Datentyp: `corrdescr`

Der Deskriptor des Korrekturgenerators.

Weitere Beispiele

Weitere Beispiele für die Instruktion CorrDiscon finden Sie unter *CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*.

Syntax

```
CorrDiscon
  [ Descr ':=' ] < variable (VAR) of corrdescr > ';'
```

Weitere Informationen

Information über	Siehe
Aufbauen einer Verbindung zu einem Korrekturgenerator	*CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*
Schreiben an einen Korrekturgenerator	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lesen der aktuellen Gesamt-Offsets	*CorrRead - Lesen der aktuellen Gesamt-Offsets auf Seite 1252*
Entfernen aller Korrekturgeneratoren	*CorrClear - Entfernen aller Korrekturgeneratoren auf Seite 176*
Korrekturdeskriptor	*corrdescr – Deskriptor des Korrekturgenerators auf Seite 1685*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.64 CorrWrite - Schreiben an einen Korrekturgenerator

Beschreibung

`CorrWrite` wird verwendet, um Offsets in das Bahn-Koordinatensystem an einen Korrekturgenerator zu schreiben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `CorrWrite`:

Beispiel 1:

```
VAR corrdescr id;
VAR pos offset;
...
CorrWrite id, offset;
```

Die aktuellen, in der Variable "offset" gespeicherten Offsets werden an den Korrekturgenerator geschrieben, der vom Deskriptor id referenziert wird.

Argumente

```
CorrWrite Descr Data
```

Descr

Datentyp: `corrdescr`

Der Deskriptor des Korrekturgenerators.

Daten

Datentyp: `pos`

Der zu schreibende Offset.

Weitere Beispiele

Weitere Beispiele für die Instruktion CorrWrite finden Sie unter *CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*.

Einschränkungen

Die besten Leistungen werden bei linearen Bahnen erzielt. Durch höhere Geschwindigkeiten und größere Winkel zwischen aufeinander folgenden linearen Bahnen vergrößert sich auch die Abweichung von der erwarteten Bahn. Dasselbe gilt für Kreise mit abnehmendem Kreisradius.

Syntax

```
CorrWrite
  [ Descr ':=' ] < variable (VAR) of corrdescr > ','
  [ Data ':=' ] < expression (IN) of pos > ';'
```

Weitere Informationen

Information über	Siehe
Aufbauen einer Verbindung zu einem Korrekturgenerator	*CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.64 CorrWrite - Schreiben an einen Korrekturgenerator

Path Offset

Fortsetzung

Information über	Siehe
Trennen der Verbindung zu einem Korrekturgenerator	*CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator auf Seite 182*
Lesen der aktuellen Gesamt-Offsets	*CorrRead - Lesen der aktuellen Gesamt-Offsets auf Seite 1252*
Entfernen aller Korrekturgeneratoren	*CorrClear - Entfernen aller Korrekturgeneratoren auf Seite 176*
Deskriptor des Korrekturgenerators	*corrdescr – Deskriptor des Korrekturgenerators auf Seite 1685*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.65 DeactEventBuffer – Deaktivierung des Ereignispuffers

Beschreibung

`DeactEventBuffer` wird verwendet, um die Verwendung des Ereignispuffers in der aktuellen Bewegungsprogrammtask zu deaktivieren.

Die Instruktionen `DeactEventBuffer` und `ActEventBuffer` sollten verwendet werden, wenn eine Anwendung mit Feinpunkten und eine fortlaufende Anwendung, bei der Signale aufgrund der langsamen Prozessausrüstung vorher gesetzt werden müssen, kombiniert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `DeactEventBuffer`:

Beispiel 1:

```
..
DeactEventBuffer;
! Use an application that use finepoints, such as SpotWelding
..
! Activate the event buffer again
ActEventBuffer;
! Now it is possible to use an application that needs
! to set signals in advance, such as Dispense
..
```

Der `DeactEventBuffer` deaktiviert den konfigurierten Ereignispuffer. Bei der Verwendung einer Anwendung mit Feinpunkten findet der Roboterstart am Feinpunkt schneller statt. Wird der Ereignispuffer mit `ActEventBuffer` aktiviert, können bei einer Anwendung mit langsamer Prozessausrüstung Signale im Voraus festgelegt werden.

Programmabarbeitung

Die Deaktivierung eines Ereignispuffers gilt jeweils für die nächste abgearbeitete Roboterinstruktion und ist so lange wirksam, bis eine `ActEventBuffer`-Instruktion ausgeführt wird.

Die Instruktion wartet mit der Deaktivierung des Ereignispuffers, bis der Roboter und die externen Achsen den Stopppunkt (`ToPoint` der aktuellen Bewegungsinstruktion) erreicht haben. Deshalb wird empfohlen, die Bewegungsinstruktion vor `DeactEventBuffer` mit einem Feinpunkt zu programmieren.

Der Standardwert (`ActEventBuffer`) wird automatisch gesetzt

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.65 DeactEventBuffer – Deaktivierung des Ereignispuffers

RobotWare - OS

Fortsetzung

- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

`DeactEventBuffer` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

Syntax

```
DeactEventBuffer ';'
```

Weitere Informationen

Information über	Siehe
Aktivierung des Ereignispuffers	*ActEventBuffer – Aktivierung des Ereignispuffers auf Seite 28*
Konfiguration von Event preset time	*Technisches Referenzhandbuch - Systemparameter*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.66 DeactUnit - Deaktivieren einer mechanischen Einheit

Verwendung

`DeactUnit` deaktiviert eine mechanische Einheit.

Mit dieser Instruktion kann auch bestimmt werden, welche Einheit aktiv ist, wenn gemeinsame Antriebseinheiten verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `DeactUnit`:

Beispiel 1:

```
DeactUnit orbit_a;
```

Deaktivieren der mechanischen Einheit `orbit_a`.

Beispiel 2:

```
MoveL p10, v100, fine, tool1;
DeactUnit track_motion;
MoveL p20, v100, z10, tool1;
MoveL p30, v100, fine, tool1;
ActUnit track_motion;
MoveL p40, v100, z10, tool1;
```

Die Einheit `track_motion` ist stationär, wenn sich der Roboter zu `p20` und `p30` bewegt. Anschließend bewegen sich Roboter und `track_motion` zu `p40`.

Beispiel 3

```
MoveL p10, v100, fine, tool1;
DeactUnit orbit1;
ActUnit orbit2;
MoveL p20, v100, z10, tool1;
```

Die Einheit `orbit1` wird deaktiviert und `orbit2` wird aktiviert.

Argumente

```
DeactUnit MechUnit
```

Mechanische Einheit

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, die deaktiviert werden soll.

Programmabarbeitung

Wenn die tatsächliche Bahn des Roboters und der externen Achsen bereit ist, wird die Bahn auf der gegenwärtigen Bahnebene gelöscht und die angegebene mechanische Einheit wird deaktiviert. Dies bedeutet, dass sie erst nach erneuter Aktivierung wieder gesteuert oder überwacht wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.66 DeactUnit - Deaktivieren einer mechanischen Einheit

RobotWare - OS

Fortsetzung

Wenn mehrere mechanische Einheiten eine gemeinsame Antriebseinheit nutzen, wird diese Einheit beim Deaktivieren einer der mechanischen Einheiten auch von der gemeinsamen Antriebseinheit getrennt.

Einschränkungen

Die Instruktion `DeactUnit` kann nicht verwendet werden, wenn eine der Achsen in der mechanischen Einheit im unabhängigen Modus ist.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden, anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`DeactUnit` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

`ActUnit` - `DeactUnit` kann auf der `StorePath`-Ebene verwendet werden, doch müssen bei der Ausführung von `RestoPath` dieselben mechanischen Einheiten wie bei der Ausführung von `StorePath` aktiviert sein. Bei einer solchen Operation sind die Bahnaufzeichnung und die Bahn auf der Basisebene nicht betroffen, doch die `StorePath`-Ebene wird gelöscht.

Syntax

```
DeactUnit
  [MechUnit ':='] < variable (VAR) of mecunit> ';'
```

Weitere Informationen

Information über	Siehe
Aktivieren mechanischer Einheiten	*ActUnit - Aktivieren einer mechanischen Einheit auf Seite 30*
Mechanische Einheiten	*mecunit – Mechanische Einheit auf Seite 1731*
Prüft, ob eine mechanische Einheit aktiviert ist.	*IsMechUnitActive - Prüfen, ob mechanische Einheit aktiv ist auf Seite 1378*
Bahnaufzeichnung	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517mecunit – Mechanische Einheit auf Seite 1731*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.67 Decr - Dekrementieren um 1

Verwendung

`Decr` wird verwendet, um 1 von einer numerischen Variablen oder Persistenten zu subtrahieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Decr`:

Siehe auch *Weitere Beispiele auf Seite 189*.

Beispiel 1:

```
Decr reg1;
```

`1` wird von `reg1` subtrahiert, d. h. `reg1:=reg1-1`.

Argumente

```
Decr Name | Dname
```

`Name`

Datentyp: `num`

Der Name der zu dekrementierenden Variablen oder Persistenten.

`Dname`

Datentyp: `dnum`

Der Name der zu dekrementierenden Variablen oder Persistenten.

Weitere Beispiele

Weitere Beispiele für die Instruktion `Decr` sind unten aufgeführt.

Beispiel 1:

```
VAR num no_of_parts:=0;
...
TPReadNum no_of_parts, "How many parts should be produced? ";
WHILE no_of_parts>0 DO
  produce_part;
  Decr no_of_parts;
ENDWHILE
```

Der Bediener wird aufgefordert, die Anzahl der zu produzierenden Teile einzugeben. Die Variable `no_of_parts` zählt, wie viele Teile noch produziert werden müssen.

Beispiel 2:

```
VAR dnum no_of_parts:=0;
...
TPReadDnum no_of_parts, "How many parts should be produced? ";
WHILE no_of_parts>0 DO
  produce_part;
  Decr no_of_parts;
ENDWHILE
```

Der Bediener wird aufgefordert, die Anzahl der zu produzierenden Teile einzugeben. Die Variable `no_of_parts` zählt, wie viele Teile noch produziert werden müssen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.67 Decr - Dekrementieren um 1

RobotWare - OS

Fortsetzung

Syntax

```
Decr
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname ':=' ] < var or pers (INOUT) of dnum >' ;'
```

Weitere Informationen

Information über	Siehe
Inkrementieren der Variablen um 1	*Incr - Inkrementieren um 1 auf Seite 260*
Subtrahieren eines Werts von einer Variablen	*Add – Addiert einen numerischen Wert auf Seite 32*
Ändern von Daten über einen beliebigen Ausdruck, z. B. Multiplikation	*":=" - Zuweisen eines Werts auf Seite 42*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.68 DropSensor - Ablegen des Objekts am Sensor

Verwendung

`DropSensor` wird verwendet, um das aktuelle Werkobjekt zu trennen. Das Programm ist dann für das nächste Werkobjekt bereit.

`DropSensor` wird zur Sensorsynchronisierung verwendet, aber nicht zur Analogsynchronisierung.

Grundlegendes Beispiel

```
MoveL *, v1000, z10, tool, \WObj:=wobj0;
SyncToSensor Ssync1\Off;
MoveL *, v1000, fine, tool, \WObj:=wobj0;
DropSensor Ssync1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
```

Argumente

```
DropSensor MechUnit
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Die sich bewegende mechanische Einheit, auf die sich die Roboterposition in der Instruktion bezieht.

Programmabarbeitung

Wenn die Verbindung zum Objekt getrennt wird, bedeutet dies, dass die Encoder-Einheit das Objekt nicht mehr verfolgt. Das Objekt wird aus der Objektqueue entfernt und kann nicht wiederhergestellt werden.

Einschränkungen

Wenn die Instruktion erteilt wird, während der Roboter das Sensorobjekt aktiv verwendet, wird die Bewegung angehalten. Die Instruktion muss erteilt werden, nachdem der Roboter das letzte synchronisierte `robtarget` passiert hat.

Die Instruktion darf nur gegeben werden, nachdem in den vorhergegangenen Bewegungsinstruktionen eine nicht synchronisierte Bewegung mit entweder einem Feinpunkt oder mehreren (> 1) Verschleifzonen verwendet wurde.

Syntax

```
DropSensor
  [ MechUnit ':='] < variable (VAR) of mecunit> ';'
```

Weitere Informationen

Information über	Siehe
Wartet auf Verbindung zu Sensor	*WaitSensor - Auf die Sensorverbindung warten auf Seite 1085*
Synchr. mit Sensor	*SyncToSensor - Synchr. mit Sensor auf Seite 875*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.68 DropSensor - Ablegen des Objekts am Sensor

Machine Synchronization

Fortsetzung

Information über	Siehe
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.69 DropWObj - Ablegen des Werkobjekts am Conveyor

Verwendung

`DropWObj` (*Drop Work Object*) wird verwendet, um die Verbindung zum aktuellen Objekt zu trennen, damit das Programm für das nächste Objekt auf dem Conveyor bereit ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `DropWObj`:

Beispiel 1:

```
MoveL *, v1000, z10, tool, \WObj:=wobj_on_cnv1;
MoveL *, v1000, fine, tool, \WObj:=wobj0;
DropWObj wobj_on_cnv1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
```

Argumente

```
DropWObj WObj
```

`WObj`

Work Object

Datentyp: `wobjdata`

Das sich bewegende Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht. Die mechanische Einheit des Conveyors muss durch die Komponente `ufmec` im Werkobjekt angegeben werden.

Programmabarbeitung

Wenn die Verbindung zum Werkobjekt getrennt wird, bedeutet dies, dass die Encoder-Einheit das Objekt nicht mehr verfolgt. Das Objekt wird aus der Objektqueue entfernt und kann nicht wiederhergestellt werden.

Einschränkungen

Wenn die Instruktion erteilt wird, während der Roboter das Conveyor-koordinierte Werkobjekt aktiv verwendet, wird die Bewegung angehalten.

Die Instruktion darf nur gegeben werden, nachdem in den vorhergegangenen Bewegungsinstruktionen ein festes Werkobjekt mit entweder einem Feinpunkt oder mehreren (> 1) Verschleifzonen verwendet wurde.

Syntax

```
DropWObj
  [ WObj ':='] < persistent (PERS) of wobjdata>';'
```

Weitere Informationen

Information über	Siehe
Warten auf Werkobjekte	*WaitWObj - Warten auf Werkobjekt am Förderer auf Seite 1106*
Conveyor Tracking	*Anwendungshandbuch - Conveyor Tracking*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.70 EOffsOff - Deaktivieren eines Offsets für zusätzliche Achsen

Verwendung

`EOffsOff` (*External Offset Off*) wird zur Deaktivierung eines Versatzes für zusätzliche Achsen verwendet.

Der Offset für zusätzliche Achsen wird durch die Instruktion `EOffsSet` oder `EOffsOn` aktiviert und gilt für alle Bewegungen, bis ein anderer Offset für zusätzliche Achsen aktiviert oder bis der Offset für zusätzliche Achsen deaktiviert wird.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `EOffsOff`:

Beispiel 1:

```
EOffsOff;
```

Deaktivierung des Offsets für zusätzliche Achsen.

Beispiel 2:

```
MoveL p10, v500, z10, tool1;
EOffsOn \ExeP:=p10, p11;
MoveL p20, v500, z10, tool1;
MoveL p30, v500, z10, tool1;
EOffsOff;
MoveL p40, v500, z10, tool1;
```

Ein Offset wird als Differenz zwischen der Position jeder Achse an `p10` und `p11` definiert. Diese Verschiebung beeinflusst die Bewegung zu `p20` und `p30`, aber nicht zu `p40`.

Programmabarbeitung

Aktive Offsets für zusätzliche Achsen werden zurückgesetzt.

Syntax

```
EOffsOff ';'
```

Weitere Informationen

Information über	Siehe
Definition eines Offsets mit zwei Positionen	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*
Definition eines Offsets mit bekannten Werten	*EOffsSet - Aktivieren eines Offsets für zusätzliche Achsen durch bekannte Werte auf Seite 198*
Deaktivieren der Programmverschiebung des Roboters	*PDispOff - Deaktivieren der Programmverschiebung auf Seite 536*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.71 EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen

Verwendung

`EOffsOn` (*External Offset On*) wird verwendet, um einen Offset für zusätzliche Achsen mithilfe von zwei Positionen zu definieren und zu aktivieren.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `EOffsOn`:

Siehe auch *Weitere Beispiele auf Seite 196*.

Beispiel 1:

```
MoveL p10, v500, z10, tool1;
EOffsOn \ExeP:=p10, p20;
```

Aktivierung eines Offsets für zusätzliche Achsen. Dies wird für jede Achse anhand der Differenz zwischen den Positionen `p10` und `p20` berechnet.

Beispiel 2:

```
MoveL p10, v500, fine \Inpos := inpos50, tool1;
EOffsOn *;
```

Aktivierung eines Offsets für zusätzliche Achsen. Da in der vorherigen Instruktion ein exakt definierter Stopppunkt verwendet wurde, kann auf das Argument `\ExeP` verzichtet werden. Die Verschiebung wird anhand der Differenz zwischen der tatsächlichen Position jeder Achse und dem programmierten Punkt (*) berechnet, der in der Instruktion gespeichert ist.

Argumente

```
EOffsOn [\ExeP] ProgPoint
```

`[ \ExeP ]`

Executed Point

Datentyp: `robtarget`

Die neue Roboterposition, die zur Berechnung des Offset verwendet wird. Wenn das Argument weggelassen wird, wird die aktuelle Achsenposition zum Zeitpunkt der Programmabarbeitung verwendet.

`ProgPoint`

Programmed Point

Datentyp: `robtarget`

Die ursprüngliche Position der Achsen zum Zeitpunkt der Programmierung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.71 EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen

RobotWare - OS

Fortsetzung

Programmabarbeitung

Der Offset wird für jede zusätzliche Achse als Differenz zwischen `\ExeP` und `ProgPoint` berechnet. Wenn `\ExeP` nicht angegeben wurde, wird die aktuelle Achsenposition zum Zeitpunkt der Programmabarbeitung verwendet Da die tatsächliche Position der Achsen verwendet wird, dürfen sich die Achsen bei der Abarbeitung von `EOffsOn` nicht bewegen.

Anhand dieses Offsets wird dann die Position der zusätzlichen Achsen in nachfolgenden Positionierinstruktionen verschoben. Er bleibt aktiv, bis ein anderer Offset aktiviert wird (mit der Instruktion `EOffsSet` oder `EOffsOn`) oder bis der Offset für zusätzliche Achsen deaktiviert wird (mit der Instruktion `EOffsOff`).

Es kann jeweils nur **ein** Offset für jede einzelne zusätzliche Achse aktiviert werden. Andererseits können mehrere `EOffsOn` nacheinander programmiert werden. In diesem Fall werden dann die verschiedenen Offsets addiert.

Der Offset der zusätzlichen Achsen wird automatisch zurückgesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `EOffsOn` sind unten aufgeführt.

Beispiel 1:

```
SearchL sen1, psearch, p10, v100, tool1;
PDispOn \ExeP:=psearch, *, tool1;
EOffsOn \ExeP:=psearch, *;
```

Ein Suchvorgang wird ausgeführt, in dem die gesuchte Position des Roboters sowie die Positionen der externen Achsen in der Position `psearch` gespeichert werden. Jede anschließende Bewegung beginnt an dieser Position durch Programmverschiebung sowohl des Roboters als auch der externen Achsen. Dies wird anhand der Differenz zwischen der gesuchten Position und dem programmierten Punkt (*) berechnet, der in der Instruktion gespeichert ist.

Syntax

```
EOffsOn
  [ '\' ExeP ':=' < expression (IN) of robtarget> ',']
  [ ProgPoint ':=' ] < expression (IN) of robtarget> ';'
```

Weitere Informationen

Information über	Siehe
Deaktivierung des Offsets für zusätzliche Achsen	*EOffsOff - Deaktivieren eines Offsets für zusätzliche Achsen auf Seite 194*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Definition eines Offsets mit bekannten Werten	*EOffsSet - Aktivieren eines Offsets für zusätzliche Achsen durch bekannte Werte auf Seite 198*
Verschiebung der Roboterbewegung	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.72 EOffsSet - Aktivieren eines Offsets für zusätzliche Achsen durch bekannte Werte

Verwendung

`EOffsSet` (*External Offset Set*) wird verwendet, um einen Offset für zusätzliche Achsen mithilfe von bekannten Werten zu definieren und zu aktivieren.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `EOffsSet`:

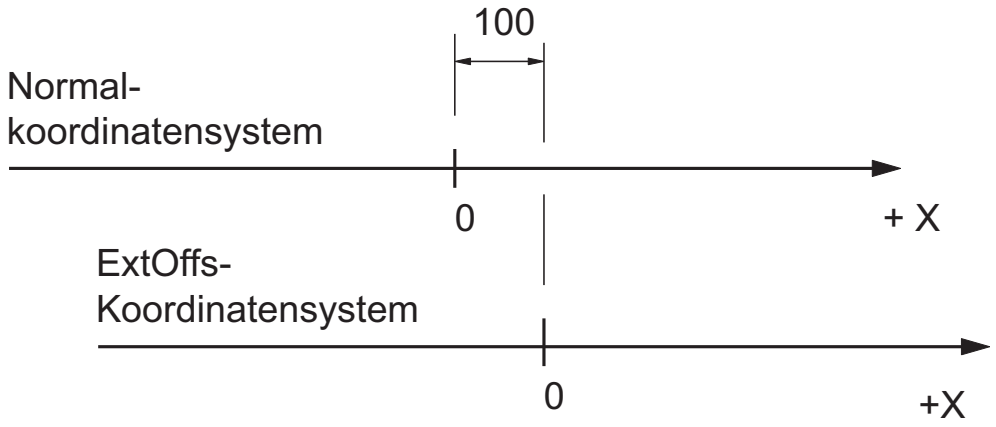
Beispiel 1:

```
VAR extjoint eax_a_p100 := [100, 0, 0, 0, 0, 0];
...
EOffsSet eax_a_p100;
```

Aktivierung eines Offsets `eax_a_p100` für zusätzliche Achsen, Bedeutung (vorausgesetzt, die logische zusätzliche Achse „a“ ist linear):

- Das `ExtOffs`-Koordinatensystem wird für die logische Achse „a“ um 100 mm verschoben (siehe Abbildung unten).
- Solange dieser Offset aktiv ist, werden alle Positionen um 100 mm in Richtung der x-Achse verschoben.

Die folgende Abbildung zeigt die Verschiebung einer zusätzlichen Achse.



xx0500002162

Argumente

```
EOffsSet EAxOffs
```

EAxOffs

External Axes Offset

Datentyp: `extjoint`

Der Offset für zusätzliche Achsen wird als Datentyp `extjoint` definiert, angegeben in:

- mm für lineare Achsen,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Grad für Rotationsachsen

Programmabarbeitung

Der Offset für externe Achsen wird aktiviert, wenn die Instruktion `EOffsSet` abgearbeitet wird, und bleibt aktiv, bis ein anderer Offset aktiviert wird (mit der Instruktion `EOffsSet` oder `EOffsOn`) oder bis der Offset für zusätzliche Achsen deaktiviert wird (mit der Instruktion `EOffsOff`).

Es kann jeweils nur ein Offset für zusätzliche Achsen aktiviert werden. Offsets können nicht miteinander durch `EOffsSet` addiert werden.

Der Offset der zusätzlichen Achsen wird automatisch zurückgesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Syntax

```
EOffsSet
  [ EAxOffs ':=' ] < expression (IN) of extjoint> ';'
```

Weitere Informationen

Information über	Siehe
Aktivieren eines Offsets für zusätzliche Achsen	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*
Deaktivierung des Offsets für zusätzliche Achsen	*EOffsOff - Deaktivieren eines Offsets für zusätzliche Achsen auf Seite 194*
Verschiebung der Roboterbewegung	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Definition von Daten des Typs „`extjoint`	*extjoint - Position von externen Achsen auf Seite 1706*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.73 EraseModule - Löschen eines Moduls

Verwendung

`EraseModule` wird verwendet, um ein Modul während der Abarbeitung aus dem Programmspeicher zu entfernen.

Es gibt keine Beschränkungen dahingehend, wie das Modul geladen wurde. Es kann manuell, von der Konfiguration oder durch eine Kombination der Instruktionen `Load`, `StartLoad` und `WaitLoad` geladen worden sein.

Das Modul kann bei der Konfiguration nicht als *Shared* definiert werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `EraseModule`:

Beispiel 1:

```
EraseModule "PART_A";
```

Löscht das Programmmodul `PART_A` aus dem Programmspeicher.

Argumente

```
EraseModule ModuleName
```

`ModuleName`

Datentyp: `string`

Der Name des zu löschenden Moduls. Beachten Sie, dass es sich hierbei um den Namen des Moduls und nicht um den Namen der Datei handelt.

Programmabarbeitung

Die Programmabarbeitung wartet das Ende des Löschvorgangs ab, bevor sie mit der nächsten Instruktion fortfährt.

Nach dem Löschen des Programmmoduls werden die übrigen Programmmodule verbunden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_MODULE`	Die Datei in der `EraseModule`-Instruktion kann nicht gelöscht werden, weil sie nicht gefunden wurde.

Einschränkungen

Es ist nicht erlaubt, ein Programmmodul zu löschen, das gerade abgearbeitet wird.

Interrupt-Routinen, E/A-Systemereignisse und andere Programmtasks können während des Löschvorgangs nicht abgearbeitet werden.

Vermeiden Sie Roboterbewegungen während des Löschvorgangs.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

En Programmstopp während der Abarbeitung der Instruktion `EraseModule`kann zu einem Sicherheitsstopp mit ausgeschalteten Motoren führen und die Fehlermeldung „20025 Stopp-Anweisung Timeout“ wird am FlexPendant angezeigt.

Syntax

```
EraseModule
[ModuleName':='']<expression (IN) of string>';'
```

Weitere Informationen

Information über	Siehe
Entfernen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*
Laden eines Programmmoduls zeitgleich mit einer anderen Programmabarbeitung	*StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802* *WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*
Akzeptieren ungelöster Verweise	*Technisches Referenzhandbuch - Systemparameter*, Abschnitt *Controller*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.74 ErrLog - Schreiben einer Fehlermeldung

Verwendung

`ErrLog` wird verwendet, um einen Fehler am FlexPendant anzuzeigen und im Meldungsprotokoll des Roboters zu speichern. Fehlernummer sowie fünf Fehlerargumente müssen angegeben werden. Die Meldung wird im Roboterprotokoll in der Prozessdomäne gespeichert. `ErrLog` kann auch zum Anzeigen von Warnungen und Informationsmeldungen verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ErrLog`:

Beispiel 1:

Wenn Sie nicht selbst eine .xml-Datei erstellen möchten, können Sie wie im folgenden Beispiel `ErrorId 4800` verwenden:

```
VAR errstr my_title := "myerror";
VAR errstr str1 := "errortext1";
VAR errstr str2 := "errortext2";
VAR errstr str3 := "errortext3";
VAR errstr str4 := "errortext4";
ErrLog 4800, my_title, str1,str2,str3,str4;
```

Auf dem FlexPendant sieht diese Meldung folgendermaßen aus:

Ereignismeldung: 4800

myerror

errortext1

errortext2

errortext3

errortext4

Beispiel 2:

Eine `ErrorId` muss in einer .xml-Datei deklariert werden. Der Wert muss zwischen 5000 und 9999 betragen. Die Fehlermeldung wird in die .xml-Datei geschrieben und die Argumente für die Meldung werden mit der Instruktion `ErrLog` gesendet. Die `ErrorId` in der .xml-Datei entspricht derjenigen in der Instruktion `ErrLog`.

HINWEIS: Bei Verwendung einer ErrorId zwischen 5000 und 9999 müssen Sie eine eigene XML-Datei installieren.

Beispiel einer Meldung in einer .xml-Datei:

```
<Message number="5210" eDefine="ERR_INPAR_RDONLY">
  <Title>Parameter error</Title>
  <Description>Task:<arg format="%s" ordinal="1" />
    <p />Symbol <arg format="%s" ordinal="2" />is read-only
    <p />Context:<arg format="%s" ordinal="3" /><p />
  </Description>
</Message>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel für eine Instruktion:

```
MODULE MyModule
  PROC main()
    VAR num errorid := 5210;
    VAR errstr arg := "P1";
    ErrLog errorid, ERRSTR_TASK, arg, ERRSTR_CONTEXT,ERRSTR_UNUSED,
         ERRSTR_UNUSED;
    ErrLog errorid \W, ERRSTR_TASK, arg,
         ERRSTR_CONTEXT,ERRSTR_UNUSED, ERRSTR_UNUSED;
  ENDPROC
ENDMODULE
```

Auf dem FlexPendant sieht diese Meldung folgendermaßen aus:

Ereignismeldung: 5210

Parameterfehler

Task: T_ROB1

Symbol P1 ist schreibgeschützt.

Kontext: MeinModul/Haupt/ErrLog

Die erste `ErrLog`-Instruktion generiert eine Fehlermeldung. Die Meldung wird im Roboterprotokoll in der Prozessdomäne gespeichert. Sie wird darüber hinaus am FlexPendant ausgegeben.

Die zweite Instruktion ist eine Warnung. Eine Meldung wird nur im Roboterprotokoll gespeichert.

Das Programm wird in beiden Fällen weiter ausgeführt, nachdem die Instruktion abgearbeitet wurde.

Argumente

```
ErrLog ErrorID [\W] | [\I] Argument1 Argument2 Argument3 Argument4
     Argument5
```

`ErrorId`

Datentyp: `num`

Die Nummer eines bestimmten Fehlers, der zu überwachen ist. Bei Verwendung einer vorinstallierten XML-Datei muss die Fehlernummer zwischen 4800 und 4814 liegen, und bei Verwendung einer eigenen XML-Datei muss sie zwischen 5000 und 9999 liegen.

`[ \W ]`

Warning

Datentyp: `switch`

Gibt eine Warnung aus, die nur im Ereignisprotokoll des Roboters gespeichert (nicht direkt am FlexPendant angezeigt) wird.

`[ \I ]`

Information

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.74 ErrLog - Schreiben einer Fehlermeldung
RobotWare - OS
Fortsetzung

Gibt eine Informationsmeldung aus, die nur im Ereignisprotokoll gespeichert (nicht direkt am FlexPendant angezeigt) wird.

Wenn keines der Argumente `\W` oder `\I` angegeben wird, generiert die Instruktion eine Fehlermeldung direkt auf dem FlexPendant und speichert sie außerdem im Ereignisprotokoll.

`Argument1`

Datentyp: `errstr`

Das erste Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument2`

Datentyp: `errstr`

Das zweite Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument3`

Datentyp: `errstr`

Das dritte Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument4`

Datentyp: `errstr`

Das vierte Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument5`

Datentyp: `errstr`

Das fünfte Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

Programmabarbeitung

Eine Fehlermeldung (max. 5 Zeilen) wird am FlexPendant angezeigt und in das Ereignisprotokoll geschrieben.

Im Fall des Arguments `\W` oder `\I` wird eine Warn- oder Informationsmeldung in das Ereignisprotokoll geschrieben.

`ErrLog` generiert Programmfehler zwischen 4800 und 4814, wenn eine vom System installierte XML-Datei verwendet wird, und Programmfehler zwischen 5000 und 9999, wenn eine eigene XML-Datei verwendet wird. Der generierte Fehler hängt von der angegebenen `ErrorID` ab.

Die Meldung wird im Ereignisprotokoll der Prozessdomäne gespeichert.

Wie eine eigene XML-Datei installiert wird, wird in *Application manual - RobotWare Add-Ins* beschrieben, siehe „Weitere Informationen“ unten.

Einschränkungen

Die Länge der Zeichenfolge (Argument1 - Argument5) ist auf 195 Zeichen beschränkt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
ErrLog
  [ErrorId ':=' ] < expression (IN) of num> ','
  [ '\'W ] | [' \' I ] ','
  [Argument1 ':=' ] < expression (IN) of errstr> ','
  [Argument2 ':=' ] < expression (IN) of errstr> ','
  [Argument3 ':=' ] < expression (IN) of errstr> ','
  [Argument4 ':=' ] < expression (IN) of errstr> ','
  [Argument5 ':=' ] < expression (IN) of errstr> ';'
```

Weitere Informationen

Information über	Siehe
Vordefinierte Daten vom Typ errstr	*errstr – Fehlerreihe auf Seite 1702*
Anzeigen von Meldungen am FlexPendant	*TPWrite - Schreiben am FlexPendant auf Seite 901* *UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Ereignisprotokoll	*Bedienungsanleitung - IRC5 mit FlexPendant*
Ereignisprotokollmeldungen, Erläuterung von XML-Datei	*Application manual - RobotWare Add-Ins*, Abschnitt *Event log messages*
So werden XML-Dateien mit Add-Ins installiert	*Application manual - RobotWare Add-Ins*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.75 ErrRaise - Schreiben einer Warnung und Aufrufen einer Fehlerbehandlung

Verwendung

`ErrRaise` wird verwendet, um einen Fehler innerhalb des Programms zu erzeugen und dann die Fehlerbehandlung der aktuellen Routine aufzurufen. In das Ereignisprotokoll wird eine Warnung geschrieben. `ErrRaise` kann auch in der Fehlerbehandlung verwendet werden, um den aktuellen Fehler an die Fehlerbehandlung der aufrufenden Routine zu übergeben.

Fehlername, Fehlernummer sowie fünf Fehlerargumente müssen angegeben werden. Die Meldung wird im Roboterprotokoll in der Prozessdomäne gespeichert.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ErrRaise`:

Beispiel 1:

Wenn Sie nicht selbst eine .xml-Datei erstellen möchten, können Sie wie im folgenden Beispiel `ErrorId 4800` verwenden:

```
MODULE MyModule
  VAR errnum ERR_BATT:=-1;
  PROC main()
    VAR num errorid := 4800;
    VAR errstr my_title := "Backup battery status";
    VAR errstr str1 := "Bacup battery is fully charged";
    BookErrNo ERR_BATT;
    ErrRaise "ERR_BATT", errorid, my_title, ERRSTR_TASK, str1,
         ERRSTR_CONTEXT,ERRSTR_EMPTY;
    ERROR
    IF ERRNO = ERR_BATT THEN
      TRYNEXT;
    ENDIF
  ENDPROC
ENDMODULE
```

Auf dem FlexPendant sieht diese Meldung folgendermaßen aus (Warnung und/oder Fehler):

Ereignismeldung: 4800

Status Reservebatterie

Task: Haupt

Reservebatterie ist vollständig aufgeladen

Kontext: MeinModul/Haupt/ErrRaise

Mithilfe der Instruktion `BookErrNo` muss eine Fehlernummer vergeben werden. Die entsprechende Zeichenfolge wird in `ErrRaise` als erstes Argument (`ErrorName`) angegeben.

`ErrRaise` erstellt einen Fehler und ruft anschließend eine Fehlerbehandlung auf. Wenn der Fehler behandelt wird, wird im Ereignisprotokoll in der Prozessdomäne eine Warnung generiert. Anderenfalls wird ein schwerwiegender Fehler generiert und das Programm wird angehalten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ErrRaise kann in einer Fehlerbehandlung auch als Unterroutine verwendet werden. In diesem Fall wird die Ausführung in der Fehlerbehandlung der aufrufenden Routine fortgesetzt.

Beispiel 2:

Eine ErrorId muss in einer .xml-Datei deklariert werden. Der Wert muss zwischen 5000 und 9999 betragen. Die Fehlermeldung wird in die .xml-Datei geschrieben und die Argumente für die Meldung werden mit der Instruktion ErrRaise gesendet. Die ErrorId in der .xml-Datei entspricht derjenigen in der Instruktion ErrRaise.

HINWEIS: Bei Verwendung einer ErrorId zwischen 5000 und 9999 müssen Sie eine eigene XML-Datei installieren.

Beispiel einer Meldung in einer .xml-Datei:

```
<Message number="7055" eDefine="SYS_ERR_ARL_INPAR_RDONLY">
  <Title>Parameter error</Title>
  <Description>Task:<arg format="%s" ordinal="1" />
    <p />Symbol <arg format="%s" ordinal="2" />is read-only
    <p />Context:<arg format="%s" ordinal="3" /><p /></Description>
</Message>
```

Beispiel für eine Instruktion:

```
MODULE MyModule
  VAR errnum ERR_BATT:=-1;
  PROC main()
    VAR num errorid := 7055;
    BookErrNo ERR_BATT;
    ErrRaise "ERR_BATT", errorid, ERRSTR_TASK,
         ERRSTR_CONTEXT,ERRSTR_UNUSED, ERRSTR_UNUSED,
         ERRSTR_UNUSED;
    ERROR
    IF ERRNO = ERR_BATT THEN
      TRYNEXT;
    ENDIF
  ENDPROC
ENDMODULE
```

Auf dem FlexPendant sieht diese Meldung folgendermaßen aus (Warnung und/oder Fehler):

Ereignismeldung: 7055

Status Reservebatterie

Task: Haupt

Reservebatterie ist vollständig aufgeladen

Kontext: MeinModul/Haupt/ErrRaise

Mithilfe der Instruktion BookErrNo muss eine Fehlernummer vergeben werden. Die entsprechende Zeichenfolge wird in ErrRaise als erstes Argument (ErrorName) angegeben.

ErrRaise erstellt einen Fehler und ruft anschließend eine Fehlerbehandlung auf. Wenn der Fehler behandelt wird, wird im Ereignisprotokoll in der Prozessdomäne eine Warnung generiert. Anderenfalls wird ein schwerwiegender Fehler generiert und das Programm wird angehalten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.75 ErrRaise - Schreiben einer Warnung und Aufrufen einer Fehlerbehandlung

RobotWare - OS

Fortsetzung

`ErrRaise` kann in einer Fehlerbehandlung auch als Unterroutine verwendet werden. In diesem Fall wird die Ausführung in der Fehlerbehandlung der aufrufenden Routine fortgesetzt.

Argumente

```
ErrRaise ErrorName ErrorId Argument1 Argument2 Argument3 Argument4
    Argument5
```

`ErrorName`

Datentyp: `string`

Mithilfe der Instruktion `BookErrNo` muss eine Fehlernummer vergeben werden. Die entsprechende Variable wird unter `ErrorName` angegeben.

`ErrorId`

Datentyp: `num`

Die Nummer eines bestimmten Fehlers, der zu überwachen ist. Bei Verwendung einer vorinstallierten XML-Datei muss die Fehlernummer zwischen 4800 und 4814 liegen, und bei Verwendung einer eigenen XML-Datei muss sie zwischen 5000 und 9999 liegen.

`Argument1`

Datentyp: `errstr`

Das erste Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument2`

Datentyp: `errstr`

Das zweite Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument3`

Datentyp: `errstr`

Das dritte Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument4`

Datentyp: `errstr`

Das vierte Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

`Argument5`

Datentyp: `errstr`

Das fünfte Argument in der Fehlermeldung. Es können beliebige Zeichenfolgen oder Daten vom Typ `errstr` verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

`ErrRaise` generiert Programmwarnungen zwischen 4800 und 4814, wenn eine vom System installierte XML-Datei verwendet wird, und Programmfehler zwischen 5000 und 9999, wenn eine eigene XML-Datei verwendet wird. Der generierte Fehler hängt von der angegebenen `ErrorID` ab Eine Warnung wird in das Roboterprotokoll in der Prozessdomäne eingetragen.

Bei der Ausführung von `ErrRaise` hängt das Verhalten davon ab, an welcher Position die Instruktion ausgeführt wird:

- Bei der Ausführung im Routinenbereich wird eine Warnung generiert, und die Ausführung wird in der Fehlerbehandlungsroutine fortgesetzt.
- Wenn die Instruktion in einer Fehlerbehandlungsroutine ausgeführt wird, wird die alte Warnung ignoriert, eine neue Warnung wird generiert und die Steuerung wird an die aufrufende Instruktion übergeben.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `ErrRaise` sind unten aufgeführt.

Beispiel 1:

```
VAR errnum ERR_BATT:=-1;
VAR errnum ERR_NEW_ERR:=-1;

PROC main()
  testerrraise;
ENDPROC

PROC testerrraise()
  BookErrNo ERR_BATT;
  BookErrNo ERR_NEW_ERR;
  ErrRaise "ERR_BATT",7055,ERRSTR_TASK,ERRSTR_CONTEXT,
       ERRSTR_UNUSED,ERRSTR_UNUSED,ERRSTR_UNUSED;
  ERROR
  IF ERRNO = ERR_BATT THEN
    ErrRaise "ERR_NEW_ERR",7156,ERRSTR_TASK,ERRSTR_CONTEXT,
         ERRSTR_UNUSED,ERRSTR_UNUSED, ERRSTR_UNUSED;
  ENDIF
ENDPROC
```

Neue Warnung 7156 aus Fehlerbehandlungsroutine generieren. Steuerung an aufrufende Routine übergeben und Abarbeitung anhalten.

Einschränkungen

Die Länge der Zeichenfolge (Argument1 - Argument5) ist auf 195 Zeichen beschränkt.

Syntax

```
ErrRaise
  [ErrorName ':=' ] < expression (IN) of string> ','
  [ErrorId ':=' ] < expression (IN) of num> ','
  [Argument1 ':=' ] < expression (IN) of errstr> ','
  [Argument2 ':=' ] < expression (IN) of errstr> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.75 ErrRaise - Schreiben einer Warnung und Aufrufen einer Fehlerbehandlung

RobotWare - OS

Fortsetzung

```
[Argument3 ':=' ] < expression (IN) of errstr> ','
[Argument4 ':=' ] < expression (IN) of errstr> ','
[Argument5 ':=' ] < expression (IN) of errstr> ';'
```

Weitere Informationen

Information über	Siehe
Vordefinierte Daten vom Typ errstr	*errstr – Fehlerreihe auf Seite 1702*
Vergeben von Fehlernummern	*BookErrNo - Vergeben einer RAPID-Systemfehlernummer auf Seite 50*
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.76 ErrWrite - Schreiben einer Fehlermeldung

Verwendung

`ErrWrite` (*Error Write*) wird verwendet, um einen Fehler am FlexPendant anzuzeigen und im Meldungsprotokoll des Roboters zu speichern. Sie kann auch zum Anzeigen von Warnungen und Informationsmeldungen verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ErrWrite`:

Beispiel 1:

```
ErrWrite "PLC error", "Fatal error in PLC" \RL2:="Call service";
Stop;
```

Eine Meldung wird im Roboterprotokoll gespeichert. Die Meldung wird auch am FlexPendant ausgegeben.

Beispiel 2:

```
ErrWrite \W, "Search error", "No hit for the first search";
RAISE try_search_again;
```

Eine Meldung wird nur im Roboterprotokoll gespeichert. Die Programmabarbeitung fährt dann fort.

Argumente

```
ErrWrite [ \W ] | [\I] Header Reason [ \RL2] [ \RL3] [ \RL4]
```

`[ \W ]`

Warning

Datentyp: `switch`

Gibt eine Warnung aus, die nur im Fehlerprotokoll des Roboters gespeichert (nicht direkt am FlexPendant angezeigt) wird.

`[ \I ]`

Information

Datentyp: `switch`

Gibt eine Informationsmeldung aus, die nur im Ereignisprotokoll gespeichert (nicht direkt am FlexPendant angezeigt) wird.

Wenn keines der Argumente `\W` oder `\I` angegeben wird, generiert die Instruktion eine Fehlermeldung direkt auf dem FlexPendant und speichert sie außerdem im Ereignisprotokoll.

`Header`

Datentyp: `string`

Überschrift der Fehlermeldung (max. 46 Zeichen).

`Reason`

Datentyp: `string`

Fehlerursache.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.76 ErrWrite - Schreiben einer Fehlermeldung

RobotWare - OS

Fortsetzung

`[ \RL2]`

Reason Line 2

Datentyp: `string`

Fehlerursache.

`[ \RL3]`

Reason Line 3

Datentyp: `string`

Fehlerursache.

`[ \RL4]`

Reason Line 4

Datentyp: `string`

Fehlerursache.

Programmabarbeitung

Eine Fehlermeldung (max. 5 Zeilen) wird am FlexPendant angezeigt und in das Meldungsprotokoll des Roboters geschrieben.

Im Fall des Arguments `\W` oder `\I` wird eine Warn- oder Informationsmeldung in das Ereignisprotokoll geschrieben.

`ErrWrite` erzeugt die Programmfehlernr. 80001 für einen Fehler, Nr. 80002 für eine Warnung (`\W`) und Nr. 80003 für eine Informationsmeldung (`\I`).

Einschränkungen

Die Gesamtlänge der Zeichenfolge (Header+Reason+\RL2+\RL3+\RL4) ist auf 195 Zeichen beschränkt.

Syntax

```
ErrWrite
  [ '\'W ] | [ '\' I ] ','
  [ Header ':=' ] < expression (IN) of string>','
  [ Reason ':=' ] < expression (IN) of string>
  [ ’\’RL2 ':=' < expression (IN) of string> ]
  [ ’\’RL3 ':=' < expression (IN) of string> ]
  [ ’\’RL4 ':=' < expression (IN) of string> ] ';'
```

Weitere Informationen

Information über	Siehe
Vordefinierte Daten vom Typ errstr	*errstr – Fehlerreihe auf Seite 1702*
Anzeigen von Meldungen am FlexPendant	*TPWrite - Schreiben am FlexPendant auf Seite 901* *UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Ereignisprotokoll	*Bedienungsanleitung - IRC5 mit FlexPendant*
Schreiben einer Fehlermeldung - Err Log	*ErrLog - Schreiben einer Fehlermeldung auf Seite 202*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.77 EXIT - Beenden der Programmabarbeitung

Verwendung

`EXIT` wird verwendet, um die Programmabarbeitung zu beenden. Der Programmneustart ist dann blockiert, d. h. das Programm kann nur ab der ersten Instruktion der Hauptroutine wieder gestartet werden.

Die Instruktion `EXIT` sollte verwendet werden, wenn schwerwiegende Fehler auftreten oder die Programmabarbeitung dauerhaft angehalten werden soll. Die Instruktion `Stop` wird verwendet, um die Programmabarbeitung temporär anzuhalten. Nach Abarbeitung der Instruktion `EXIT` ist der Programmzeiger nicht mehr vorhanden. Um die Programmabarbeitung fortzusetzen, muss der Programmzeiger gesetzt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `EXIT`:

Beispiel 1:

```
ErrWrite "Fatal error","Illegal state";
EXIT;
```

Die Programmabarbeitung stoppt und kann ab dieser Position im Programm nicht neu gestartet werden.

Syntax

```
EXIT ';'
```

Weitere Informationen

Information über	Siehe
Temporäres Stoppen der Programmabarbeitung	*Stop - Stoppt die Programmabarbeitung auf Seite 833*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.78 ExitCycle - Abbrechen des aktuellen und Starten des nächsten Zyklus

Verwendung

`ExitCycle` wird verwendet, um den aktuellen Zyklus abzubrechen und den Programmzeiger zurück auf die erste Instruktion in der Hauptroutine zu setzen.

Wenn das Programm im kontinuierlichen Modus ausgeführt wird, beginnt es mit der Abarbeitung des nächsten Zyklus.

Wenn die Abarbeitung im zyklischen Modus erfolgt, stoppt sie bei der ersten Instruktion in der Hauptroutine.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ExitCycle`:

Beispiel 1:

```
VAR num cyclecount:=0;
VAR intnum error_intno;

PROC main()
  IF cyclecount = 0 THEN
    CONNECT error_intno WITH error_trap;
    ISignalDI di_error,1,error_intno;
  ENDIF
  cyclecount:=cyclecount+1;
  ! start to do something intelligent
  ...
ENDPROC

TRAP error_trap
  TPWrite "ERROR, I will start on the next item";
  ExitCycle;
ENDTRAP
```

Damit wird der nächste Zyklus gestartet, wenn das Signal `di_error` gesetzt ist.

Programmabarbeitung

Die Abarbeitung von `ExitCycle` in einer mechanische Einheiten steuernden Programmtask führt in der aktuellen Task zu folgendem Ergebnis:

- Laufende Roboterbewegung stoppt.
- Alle Roboterbahnen, die nicht auf allen Bahnebenen ausgeführt werden (normale Ebene und `StorePath` -Ebene), werden gelöscht.
- Alle Instruktionen, die gestartet, aber nicht auf allen Abarbeitungsebenen beendet wurden, (normale Ebene und `TRAP`-Ebene) werden unterbrochen.
- Der Programmzeiger wird an die erste Instruktion in der Hauptroutine versetzt.
- Die Programmabarbeitung fährt mit dem nächsten Zyklus fort.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.78 ExitCycle - Abbrechen des aktuellen und Starten des nächsten Zyklus

RobotWare - OS

Fortsetzung

Die Abarbeitung von `ExitCycle` in einer anderen Programmtask, die keine mechanischen Einheiten steuert, führt in der aktuellen Task zu folgendem Ergebnis:

- Alle Instruktionen, die gestartet, aber nicht auf allen Abarbeitungsebenen beendet wurden, (normale Ebene und `TRAP`-Ebene) werden unterbrochen.
- Der Programmzeiger wird an die erste Instruktion in der Hauptroutine versetzt.
- Die Programmabarbeitung fährt mit dem nächsten Zyklus fort.

Alle anderen modalen Einstellungen im Programm und System werden **nicht** durch `ExitCycle` beeinflusst, z. B.:

- Der aktuelle Wert von Variablen oder Persistenten.
- Jegliche Bewegungseinstellungen, wie z. B. die Sequenz `StorePath-RestoPath` , Weltzonen usw.
- Geöffnete Dateien, Verzeichnisse usw.
- Definierte Interrupte usw.

Wenn bei Verwendung von `ExitCycle` in Routineaufrufen die Einstiegsroutine mit „Move PP to Routine ...“ oder „Call Routine ...“ definiert ist, unterbricht `ExitCycle` den aktuellen Zyklus und versetzt den Programmzeiger zurück an die erste Instruktion in der Einstiegsroutine (anstelle der Hauptroutine, wie oben angegeben).

Syntax

```
ExitCycle';'
```

Weitere Informationen

Information über	Siehe
Stoppen nach einem schwerwiegenden Fehler	*EXIT - Beenden der Programmabarbeitung auf Seite 213*
Beenden der Programmabarbeitung	*EXIT - Beenden der Programmabarbeitung auf Seite 213*
Stoppen für Programmaktionen	*Stop - Stoppt die Programmabarbeitung auf Seite 833*
Beenden der Abarbeitung einer Routine	*RETURN - Beenden der Abarbeitung einer Routine auf Seite 624*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.79 FitCircle – Passt einen Kreis an 3D-Punkte an

Verwendung

`FitCircle` wird verwendet, um einen Kreis an eine Reihe von 3D-Punkten anzupassen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `FitCircle`.

Beispiel 1:

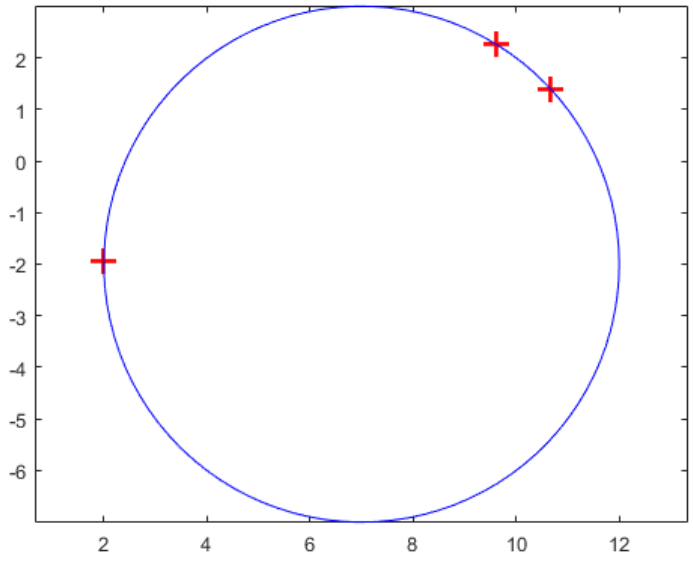
```
VAR pos points{3}:=[ [2.000264140454799, -1.948606082287765, 3],
     [10.666326255802462, 1.399713485871053, 3],
     [9.609499187363362, 2.265033879249959, 3]];
VAR num radius;
VAR pos center;
VAR pos normal;
FitCircle points, center, radius, normal;
```

Bei Angabe von nur drei Punkten errechnet `FitCircle` einen Kreis, der genau durch jeden einzelnen Punkt verläuft. In diesem Beispiel entsteht folgender Kreis:

```
center = [7,-2,3]
radius = 5
normal = [0,0,1]
```

In diesem einfachen Beispiel liegen alle 3D-Punkte auf derselben Z-Koordinate. Der identifizierte Kreis muss daher auf der XY-Ebene liegen. Im Allgemeinen identifiziert die Methode die Ebene, auf der sich der Kreis befindet. Die Ebene wird durch die errechnete Normale festgelegt und besteht aus einem Einheitsvektor, rechtwinklig zum Kreis.

Der Kreis und die Eingabepunkte werden in der folgenden Abbildung illustriert.



xx1700000731

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 2:

```
VAR pos points{10}:=[
[-7.2, -5.254055558738700, 3.307061712883007],
[-7.2, -3.211963764808295, 3.090792209685409],
[-7.2, -5.650784519354138, 4.074184087507510],
[-7.2, -4.261241341238363, 3.549808031134106],
[-7.2, -3.780658372123544, 12.570750298513245],
[-7.2, 1.309442476421255, 11.013856026376601],
[-7.2, -4.649041803426594, 3.435039251520052],
[-7.2, 1.403058916454365, 6.576147932013719],
[-7.2, -7.546395053424201, 9.841667138860654],
[-7.2, -2.490309697793828, 2.788705869634919]];
VAR pos center;
VAR num radius;
VAR pos normal;
VAR num rms;
VAR num maxErr;
VAR num indexWorst;

FitCircle points, center, radius, normal \RMS:=rms \MaxErr:=maxErr
     \IndexWorst:=indexWorst;
```

In diesem Fall wird der Kreis an zehn Punkte angepasst, die nicht auf dem Kreis liegen. Das Ergebnis ist ein Kreis, der anhand der Methode der kleinsten Quadrate an die Punkte angepasst wurde. Zur Vereinfachung des Beispiels befinden sich alle Punkte auf einer Ebene, die parallel zur YZ-Ebene verläuft.

Der resultierende Kreis ist:

```
center = [-7.2,-2.92489,7.96317]
radius = 4.88656
normal = [1,0,0]
```

Die abweichenden Parameter der Anpassung sind:

```
rms = 0.2387
maxErr = 0.3418
indexWorst = 8
```

Fortsetzung auf nächster Seite

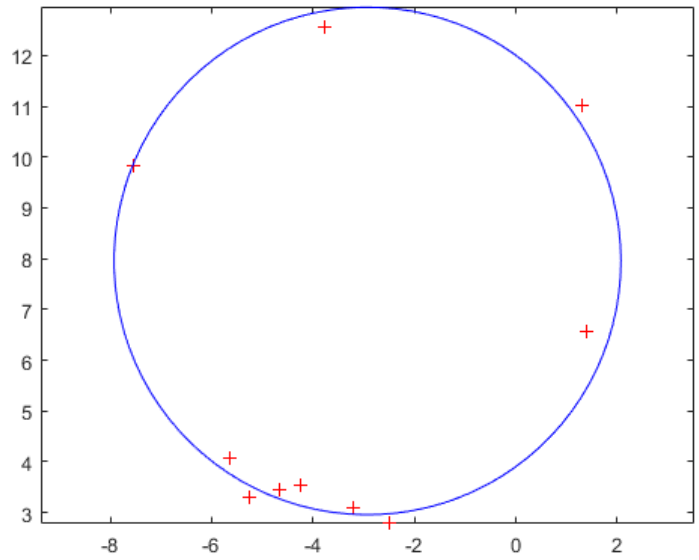
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.79 FitCircle – Passt einen Kreis an 3D-Punkte an

RobotWare - OS

Fortsetzung

Der Kreis und die Eingabepunkte werden in der folgenden Abbildung illustriert.



xx1700000732

Argumente

```
FitCircle Points [\NumPoints] Center Radius Normal [\RMS] [\MaxErr]
    [\IndexWorst]
```

`Points`

Datentyp: array of pos

`Points` ist eine Reihe von 3D-Punkten für die Kreisanpassung.

`[\NumPoints]`

Datentyp: `num`

Mit dem optionalen Argument `NumPoints` lässt sich festlegen, wie viele der Punkte verwendet werden sollen. Bei Nichtangabe werden alle Punkte der Reihe `Points` verwendet.

`Center`

Datentyp: `pos`

Der Mittelpunkt des resultierenden Kreises.

`Radius`

Datentyp: `num`

Der Radius des resultierenden Kreises.

`Normal`

Datentyp: `pos`

Ein Einheitslängenvektor, der sich rechtwinklig zur Ebene des identifizierten Kreises befindet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\RMS]

Datentyp: num

Optionales Argument, das die Abweichung des quadratischen Mittels des angepassten Kreises enthält.

[\MaxErr]

Datentyp: num

Optionales Argument, das den Höchstabstand zwischen dem resultierenden Kreis und den Eingabepunkten enthält.

[\IndexWorst]

Datentyp: num

Optionales Argument, das den Index jenes Punktes enthält, welcher den Höchstabstand zum Kreis aufweist.

Programmabarbeitung

CircleFit passt einen Kreis an eine Reihe von 3D-Punkten an.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_ARRAY_SIZE	Falsche Dimensionen oder falsche Werte für optionale Argumente werden verwendet.

Einschränkungen

Die Anzahl der 3D-Punkte muss zwischen 3 und 100 liegen.

Syntax

```
CircleFit
  [ Points ':='] <array {*} expression (IN) of dnum> ','
  [ '\' NumPoints ':=' < expression (IN) of num>] ','
  [ Center ':='] <variable (VAR) of pos> ','
  [ Radius ':='] <variable (VAR) of num> ','
  [ Normal ':='] <variable (VAR) of pos>
  [ '\' RMS ':=' <variable (VAR) of num>]
  [ '\' MaxErr ':=' <variable (VAR) of num> ]
  [ '\' IndexWorst ':=' <variable (VAR) of num> ] ';'
```

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen.	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.80 FOR - Vorgegebene Anzahl an Wiederholungen

Verwendung

FOR wird verwendet, wenn eine oder mehrere Instruktionen mit einer bestimmten Anzahl von Wiederholungen ausgeführt werden sollen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion FOR:

Siehe auch *Weitere Beispiele auf Seite 221*.

Beispiel 1:

```
FOR i FROM 1 TO 10 DO
  routine1;
ENDFOR
```

Wiederholt die Prozedur routine110-mal.

Argumente

```
FOR Loop counter FROM Start value TO End value [STEP Step value]
     DO ... ENDFOR
```

`Loop counter`

Identifier

Der Name der Information, die den Wert des aktuellen Schleifenzählers enthalten wird. Die Daten werden automatisch deklariert.

Wenn der Name des Schleifenzählers mit dem Namen bereits vorhandener Daten im aktuellen Gültigkeitsbereich übereinstimmt, werden die bestehenden Daten in der FOR-Schleife verborgen und nicht beeinflusst.

`Start value`

Datentyp: Num

Der gewünschte Anfangswert des Schleifenzählers (gewöhnlich ganzzahlige Werte).

`End value`

Datentyp: Num

Der gewünschte Endwert des Schleifenzählers (gewöhnlich ganzzahlige Werte).

`Step value`

Datentyp: Num

Der Schrittwert, um den der Schleifenzähler in jeder Schleife inkrementiert (oder dekrementiert) wird (gewöhnlich ganzzahlige Werte).

Wenn dieser Wert nicht angegeben wird, wird der Schrittwert automatisch auf 1 gesetzt (oder auf -1, falls der Startwert größer als der Endwert ist).

Programmabarbeitung

1. Die Ausdrücke für den Start-, End- und Schrittwert werden berechnet.
2. Dem Schleifenzähler wird der Startwert zugewiesen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3 Der Wert des Schleifenzählers wird überprüft, um festzustellen, ob er zwischen dem Start- und Endwert liegt oder ob er gleich dem Start- oder Endwert ist. Wenn der Wert des Schleifenzählers außerhalb dieses Bereichs liegt, stoppt die FOR-Schleife und die Programmabarbeitung fährt mit der Instruktion fort, die auf ENDFOR folgt.
4 Die Instruktionen in der FOR-Schleife werden abgearbeitet.
5 Der Schleifenzähler wird mit dem Schrittwert inkrementiert (oder dekrementiert).
6 Die FOR-Schleife wird beginnend ab Punkt 3 wiederholt.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion FOR sind unten aufgeführt.

Beispiel 1:

```
FOR i FROM 10 TO 2 STEP -2 DO
  a{i} := a{i-1};
ENDFOR
```

Die Werte in einem Datenfeld werden wie folgt nach oben angepasst: a{10}:=a{9}, a{8}:=a{7} usw.

Einschränkungen

Auf den Schleifenzähler (Datentyp num) kann nur aus der FOR-Schleife zugegriffen werden; daher verbirgt er andere Daten und Routinen, die denselben Namen haben. Er kann von den Instruktionen in der FOR-Schleife nur gelesen (nicht aktualisiert) werden.

Dezimalwerte für Start-, End- oder Schrittwerte können in Kombination mit exakten Abschlussbedingungen für die FOR-Schleife nicht verwendet werden (undefiniert, unabhängig davon, ob die letzte Schleife abgearbeitet wird).

Anmerkungen

Wenn die Wiederholungen so häufig erfolgen sollen, wie ein vorgegebener Ausdruck als TRUE berechnet wird, sollte stattdessen die Instruktion WHILE verwendet werden.

Syntax

```
FOR <loop variable> FROM <expression> TO <expression>
  [ STEP <expression> ] DO
  <statement list>
ENDFOR
```

Weitere Informationen

Information über	Siehe
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*
So lange wiederholen bis...	*WHILE – So lange wiederholen bis... auf Seite 1110*
Bezeichner	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.81 FricIdInit - Initiierung der Reibungsidentifikation

Verwendung

`FricIdInit` markiert den Startpunkt einer Sequenz von Bewegungsinstruktionen, die wiederholt werden, um die interne Reibung der Roboter zu berechnen.

Beispiel

Ein grundlegendes Beispiel für die Verwendung einer kreisförmigen Bewegung zur Berechnung der internen Reibung des Roboters für diese Bewegung:

```
PERS num friction_levels{6};

! Start of the friction calculation sequence
FricIdInit;

! Execute the move sequence
MoveC p10, p20, Speed, z0, Tool;
MoveC p30, p40, Speed, z0, Tool;

! Repeat the sequence and calculate the friction
FricIdEvaluate friction_levels;

! Activate compensation for the calculated friction levels
FricIdSetFricLevels friction_levels;
```

Voraussetzungen

Der Systemparameter *Friction FFW On* muss auf TRUE gesetzt werden. Anderenfalls hat die Instruktion `FricIdInit` keine Wirkung.

Einschränkungen

- `FricIdInit` kann nur bei TCP-Robotern verwendet werden.
- Kann nur von Bewegungstasks aus ausgeführt werden.
- Der Roboter muss sich auf der Basispfadebene bewegen.
- Die Reibungsabstimmung kann nicht mit dem Synchronbewegungsmodus kombiniert werden. Das heißt, `SyncMoveOn` ist zwischen `FricIdInit` und `FricIdEvaluate` nicht erlaubt.
- Die Folge von Bewegungen, für die eine Reibungsabstimmung durchgeführt wird, muss mit einem Feinpunkt beginnen und enden. Wenn nicht, werden die Feinpunkte während des Abstimmungsvorgangs eingefügt.

Syntax

```
FricIdInit ';'
```

Weitere Informationen

Information über	Siehe
Advanced robot motion	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.82 FricIdEvaluate - Evaluierung der Reibungsidentifikation

Verwendung

Durch `FricIdEvaluate` wiederholt der Roboter die Bewegung zwischen den Instruktionen `FricIdInit` und `FricIdEvaluate` während der Berechnung der Reibung für jede Roboterachse.

Beispiel

Ein grundlegendes Beispiel für die Verwendung einer kreisförmigen Bewegung zur Berechnung der internen Reibung des Roboters für diese Bewegung:

```
PERS num friction_levels{6};

! Start of the friction calculation sequence
FricIdInit;

! Execute the move sequence
MoveC p10, p20, Speed, z0, Tool;
MoveC p30, p40, Speed, z0, Tool;

! Repeat the sequence and calculate the friction
FricIdEvaluate friction_levels;

! Activate compensation for the calculated friction levels
FricIdSetFricLevels friction_levels;
```

Argumente

```
FricIdEvaluate FricLevels [\MechUnit] [\BwdSpeed] [\NoPrint]
     [\FricLevelMax] [\FricLevelMin] [\OptTolerance]
```

`FricLevels`

Friction levels

Datentyp: `array of num`

Wenn `FricIdEvaluate` abgeschlossen ist, enthält das Datenfeld `FricLevels` die abgestimmten Reibungsgrößen für alle Roboterachsen. Dieses Datenfeld muss so deklariert sein, dass es so viele Elemente besitzt wie der Roboter Achsen besitzt. Beachten Sie, dass die Instruktion `FricIdSetFricLevels` aufgerufen werden muss, damit diese Werte eine Wirkung haben.

`[\MechUnit]`

Mechanical unit

Datentyp: `mecunit`

Das Argument `MechUnit` ist optional. Wenn es ausgelassen wird, erfolgt die Reibungsabstimmung für die mechanische Einheit, die durch die vordefinierte RAPID-Variable `ROB_ID` dargestellt wird, die eine Referenz für den TCP-Roboter in der aktuellen Programmtask ist. Die Reibungskompensation kann nur bei TCP-Robotern verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.82 FricIdEvaluate - Evaluierung der Reibungsidentifikation
Advanced Shape Tuning
Fortsetzung

[\BwdSpeed]

Backward speed

Datentyp: speeddata

Nach jeder Wiederholung des Abstimmungsprozesses bewegt sich der Roboter auf der programmierten Bahn zurück. Die Rückwärtsbewegung wird standardmäßig in der programmierten Geschwindigkeit durchgeführt. Um den Prozess zu beschleunigen, kann das optionale Argument BwdSpeed verwendet werden, um während der Rückwärtsbewegung eine höhere Geschwindigkeit vorzugeben. Dies hat *keinen* Einfluss auf das Abstimmungsergebnis.

[\NoPrint]

Datentyp: switch

Wenn das Argument NoPrint verwendet wird, wird am FlexPendant kein Text über den Fortschritt der Wiederholungen zur Identifikation der Reibung geschrieben.

[\FricLevelMax]

Friction level max

Datentyp: num

Normalerweise wird der optimale Reibungswert gefunden, indem man Werte zwischen 1% und 500% des konfigurierten Reibungswerts ausprobiert. In seltenen Fällen kann dies eine Fehlermeldung erzeugen (Achsgeschwindigkeitsfehler). Um dies zu vermeiden, verwenden Sie das Argument FricLevelMax und setzen Sie es auf einen Wert unter 500. Wenn beispielsweise FricLevelMax auf 400 gesetzt wird, werden Werte zwischen 1% und 400% getestet.

Es sind Werte zwischen 101 und 500 zulässig.

[\FricLevelMin]

Friction level min

Datentyp: num

Normalerweise wird der optimale Reibungswert gefunden, indem man Werte zwischen 1 % und 500 % des konfigurierten Reibungswerts ausprobiert. Um einen größeren Startwert als 1 % einzustellen, verwenden Sie das Argument FricLevelMin. Wenn z. B. FricLevelMin auf 80 gesetzt wird, werden Werte zwischen 80 % und 500 % getestet.

Es sind Werte zwischen 1 und 99 zulässig.

[\OptTolerance]

Optimization tolerance

Datentyp: num

Normalerweise wird der optimale Reibungswert gefunden, indem man so lange Werte ausprobiert, bis eine geringe Toleranz gefunden wird. Um den Vorgang zu beschleunigen, kann dieser Wert vergrößert werden. Durch die Vergrößerung des Werts wird eventuell eine weniger genaues Ergebnis erzielt.

Es sind Werte zwischen 1 und 10 zulässig. Der Standardwert ist 1.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_FRICTUNE_FATAL	Ein Fehler tritt bei der Reibungsabstimmung auf.

Hinweis

Wenn die Reibungsabstimmung keinerlei Auswirkung hat, stellen Sie sicher, dass der Systemparameter *Friction FFW On* auf TRUE gesetzt ist.

Voraussetzungen

Der Systemparameter *Friction FFW On* muss auf TRUE gesetzt werden. Anderenfalls hat die Instruktion FricIdEvaluate keine Wirkung.

Einschränkungen

- FricIdEvaluate kann nur bei TCP-Robotern verwendet werden.
- FricIdEvaluate kann nur von Bewegungstasks aus ausgeführt werden.
- Der Roboter muss sich auf der Basispfadebene bewegen.
- Für ein MultiMove-System kann die Achsenreibungsabstimmung jeweils nur an einem Roboter durchgeführt werden. Mehrere Roboter können FricIdEvaluate gleichzeitig ausführen, aber sie stehen automatisch still und warten auf ihren Einsatz, solange ein anderer Roboter mit der Reibungsabstimmung beschäftigt ist.
- Die Reibungsabstimmung kann nicht mit dem Synchronbewegungsmodus kombiniert werden. Das heißt, SyncMoveOn ist zwischen FricIdInit und FricIdEvaluate nicht erlaubt.
- Die Folge von Bewegungen, für die eine Reibungsabstimmung durchgeführt wird, muss mit einem Feinpunkt beginnen und enden. Wenn nicht, werden die Feinpunkte während des Abstimmungsvorgangs eingefügt.

Syntax

```
FricIdEvaluate
  [ FricLevels ':='] < persistent array {*} (PERS) of num >
  ['\' MechUnit ':=' < variable (VAR) of mecunit >]
  ['\' BwdSpeed ':=' < expression (IN) of speeddata >]
  ['\' NoPrint]
  ['\' FricLevelMax ':=' < expression (VAR) of num >]
  ['\' FricLevelMin ':=' < expression (VAR) of num >]
  ['\' OptTolerance ':=' < expression (VAR) of num >] ';'
```

Weitere Informationen

Information über	Siehe
Advanced robot motion	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.83 FricIdSetFricLevels - Stellt Reibungslevel nach Reibungsidentifikation ein

Verwendung

`FricIdSetFricLevels` wird auf die Reibungsgröße für jede Roboterachse einer mechanischen Einheit gesetzt.

Beispiel

Ein grundlegendes Beispiel für die Verwendung einer kreisförmigen Bewegung zur Berechnung der internen Reibung des Roboters für diese Bewegung:

```
PERS num friction_levels{6};

! Start of the friction calculation sequence
FricIdInit;

! Execute the move sequence
MoveC p10, p20, Speed, z0, Tool;
MoveC p30, p40, Speed, z0, Tool;

! Repeat the sequence and calculate the friction
FricIdEvaluate friction_levels;

! Activate compensation for the calculated friction levels
FricIdSetFricLevels friction_levels;
```

Argumente

```
FricIdSetFricLevels FricLevels [\MechUnit]
```

`FricLevels`

Friction levels

Datentyp: `array of num`

Das Datenfeld `FricLevels` legt die Reibungsgröße für jede Achse in Prozent von der Standardreibung fest. Die Werte müssen im Intervall 0-500 liegen.

`[\MechUnit]`

Mechanical unit

Datentyp: `mecunit`

Das Argument `MechUnit` ist optional. Wenn es ausgelassen wird, werden die Reibungsgrößen für die mechanische Einheit gesetzt, die durch die vordefinierte RAPID-Variable `ROB_ID` dargestellt wird. Die Reibungskompensation kann nur bei TCP-Robotern verwendet werden.

Programmabarbeitung

Die Einstellungen der Reibungsgrößen bleiben aktiv, bis:

- Die Programmausführung vom Beginn gestartet wird (PZ auf Main)
- Ein weiterer Aufruf zu `FricIdSetFricLevels` durchgeführt wird
- Ein neues Programm geladen wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Die Steuerung wird mit dem Neustartmodus **System zurücksetzen** neu gestartet.

Voraussetzungen

Der Systemparameter *Friction FFW On* muss auf TRUE gesetzt werden. Anderenfalls hat die Instruktion `FricIdSetFricLevels` keine Wirkung.

Einschränkungen

- `FricIdSetFricLevels` kann nur bei TCP-Robotern verwendet werden.

Syntax

```
FricIdSetFricLevels
  [ FricLevels ':='] < array {*} (IN) of num >
  ['\' MechUnit ':=' < variable (VAR) of mecunit >] ';'
```

Weitere Informationen

Information über	Siehe
Advanced robot motion	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.84 GetDataVal - Abrufen des Werts eines Datenobjekts

Verwendung

`GetDataVal` (*Get Data Value*) ruft einen Wert eines Datenobjekts ab, das mit einer Zeichenfolgenvariablen angegeben ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `GetDataVal`:

Beispiel 1:

```
VAR datapos block;
VAR string name;
VAR num valuevar;
...
SetDataSearch "num" \Object:="my.*" \InMod:="mymod";
WHILE GetNextSym(name,block) DO
  GetDataVal name\Block:=block,valuevar;
  TPWrite name+" "\Num:=valuevar;
ENDWHILE
```

Diese Sitzung druckt auf dem FlexPendant alle Variablen des Typs `num`, die mit `my` beginnen und im Modul `mymod` enthalten sind, zusammen mit ihrem Wert aus.

Beispiel 2:

```
VAR num NumArrConst_copy{2};
...
GetDataVal "NumArrConst", NumArrConst_copy;
TPWrite "Pos1 = " \Num:=NumArrConst_copy{1};
TPWrite "Pos2 = " \Num:=NumArrConst_copy{2};
```

Dieser Vorgang druckt die `num`-Variablen in das Datenfeld `NumArrConst`.

Argumente

```
GetDataVal Object [\Block]|[\TaskRef]|[\TaskName] Value
```

`Object`

Datentyp: `string`

Name des Datenobjekts

`[ \Block ]`

Datentyp: `datapos`

Der eingeschlossene Block für das Datenobjekt. Kann nur mit der Funktion `GetNextSym` abgerufen werden.

Falls dieses Argument weggelassen wird, wird der Wert des sichtbaren Datenobjekts im derzeitigen Gültigkeitsbereich der Programmabarbeitung abgerufen.

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, in der das angegebene Datenobjekt gesucht werden soll. Bei Verwendung dieses Arguments können Sie die `PERS`- oder

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

TASKPERS-Deklaration in anderen Tasks suchen. Jede andere Deklaration führt zu einem Fehler.

Für alle Programmtasks im System stehen die vordefinierten Variablen des Datentyps taskid zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“; für den Task T_ROB1 lautet die Variablenidentität z. B. T_ROB1Id.

[\TaskName]

Datentyp: string

Der Name der Programmtask, in der das angegebene Datenobjekt gesucht werden soll. Bei Verwendung dieses Arguments können Sie die PERS- oder TASKPERS-Deklaration in anderen Tasks suchen. Jede andere Deklaration führt zu einem Fehler.

Value

Datentyp: anytype

Variable zum Speichern des abgerufenen Werts. Der Datentyp muss dem Datentyp des zu suchenden Datenobjekts entsprechen. Der Wert kann aus einer Konstanten, einer Variablen oder einer Persistenten abgerufen werden. Er muss jedoch in einer Variablen gespeichert werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_SYM_ACCESS	• Das Datenobjekt existiert nicht. • Das Datenobjekt besteht aus Routinedaten oder Routineparametern und befindet sich nicht in der aktuellen aktiven Routine. • In anderen Tasks wird nach anderen Deklarationen als PERS oder TASK PERS gesucht.
ERR_INVDIM	Das Datenobjekt und die im Argument Value verwendete Variable weisen unterschiedliche Dimensionen auf.
ERR_SYMBOL_TYPE	Das Datenobjekt und die im Argument Value verwendete Variable weisen unterschiedliche Typen auf. Wenn ALIAS-Datentypen verwendet werden, erhalten Sie ebenfalls diesen FEHLER, auch wenn die Typen den gleichen Basisdatentyp aufweisen.
ERR_TASKNAME	Wenn der Name der Programmtask im Argument \TaskName im System nicht gefunden werden kann, wird die Systemvariable ERRNO auf ERR_TASKNAME gesetzt.

Bei Verwendung der Argumente TaskRef oder TaskName können Sie die PERS oder TASK PERS-Deklarationen in anderen Tasks suchen. Andere Deklarationen führen zu einem Fehler, und die Systemvariable ERRNO wird auf ERR_SYM_ACCESS gesetzt. Das Suchen nach einem als PERS deklarierten LOCAL in anderen Tasks führt ebenfalls zu einem Fehler und zum Setzen der Systemvariablen ERRNO auf ERR_SYM_ACCESS.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.84 GetDataVal - Abrufen des Werts eines Datenobjekts

RobotWare - OS

Fortsetzung

Einschränkungen

Bei dem Datentyp semivalue ist eine Suche nach dem zugehörigen value-Datentyp nicht möglich. Wenn z. B. nach `dionum` gesucht wird, werden keine Treffer für `signaldi`-Signale erzielt, und bei der Suche nach `num` werden keine Treffer für `signalgi`-Signale oder `signalai`-Signale erzielt.

Es ist nicht möglich, den Wert einer Variablen abzurufen, die in einem integrierten RAPID-Modul als `LOCAL` deklariert wurde.

Syntax

```
GetDataVal
  [ Object ':=' ] < expression (IN) of string >
  ['\'Block' :='<variable (VAR) of datapos>]
  |[ '\'TaskRef' :=' <variable (VAR) of taskid>]
  |[ '\'TaskName' :=' <expression (IN) of string>] ',']
  [ Value ':=' ] <variable (VAR) of anytype>]';'
```

Weitere Informationen

Information über	Siehe
Definieren eines Symbols, das in einem Suchvorgang gesetzt wird	*SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird auf Seite 702*
Abrufen des nächsten passenden Symbols	*GetNextSym - Abrufen des nächsten passenden Symbols auf Seite 1328*
Setzen des Werts eines Datenobjekts	*SetDataVal - Setzen des Werts eines Datenobjekts auf Seite 707*
Setzen des Werts vieler Datenobjekte	*SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge auf Seite 697*
Zugehöriger Datentyp "datapos"	*datapos – Umschließender Block für ein Datenobjekt auf Seite 1687*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.85 GetGroupSignalInfo - Lesen der Informationen zu dem digitalen Gruppensignal

Verwendung

`GetGroupSignalInfo` wird zum Auslesen der Informationen über ein Gruppensignal von E/A verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `GetGroupSignalInfo`.

Beispiel 1:

```
VAR dnum DValue;
VAR string label;
VAR string devicename;

GetGroupSignalInfo go1 \MaxDValue:=DValue \Label:=label
     \DeviceName:=devicename;
```

Auslesen des maximalen Signalwerts, der auf das Signal go1, das Signal Identification Label und den Gerätenamen, dem das Signal zugeordnet ist, eingestellt werden kann.

Argumente

```
GetGroupSignalInfo Signal [\MaxDValue] [\Label] [\DeviceName]
```

`Signal`

Datentyp: `signalxx`

Der Signalbezeichner gemäß dem Programm (Datentyp signalgo oder signalgi) über die Informationen abgerufen werden sollen.

`\MaxDValue`

Datentyp: `dnum`

MaxDValue ist der Höchstwert, der auf das Signal eingestellt werden kann. Es basiert auf der Gerätezuordnung des Signals.

`\Label`

Datentyp: `string`

Das Label ist das angegebene Signal Identification Label.

`\DeviceName`

Datentyp: `string`

DeviceName ist das Gerät, dem das Signal zugewiesen ist.

Programmabarbeitung

Die Instruktion liest die Informationen über ein Gruppensignal von E/A aus.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.85 GetGroupSignalInfo - Lesen der Informationen zu dem digitalen Gruppensignal

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.

Syntax

```
GetGroupSignalInfo
  [ Signal ':=' ] < variable (VAR) of anytype >
  [ '\' MaxDvalue ':=' < variable (VAR) of dnum >]
  [ '\' Label ':=' < variable (VAR) of string >]
  [ '\' DeviceName ':=' < variable (VAR) of string >] ';'
```

Weitere Informationen

Information über	Siehe
Leseattribut eines Systemparameters	*ReadCfgData - Lesen von Attributen eines Systemparameters auf Seite 586*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.86 GetJoinData – Abrufen der gelenkspezifischen Daten

Verwendung

`GetJointData` wird verwendet, um gelenkspezifische Daten von einer angegebenen mechanischen Einheit abzulesen. Die Informationen, die für die angegebene Achse abgelesen werden können, sind Position, Geschwindigkeit, Drehmoment und geschätztes, externes Drehmoment.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `GetJointData`.

Beispiel 1:

```
VAR num position;
VAR num speed;
VAR num torque;
VAR num exttorque;
...
GetJointData \MechUnit:=ROB_1, 1 \Position:=position \Speed:=speed
     \Torque:=torque \ExtTorque:=exttorque;
```

Die aktuelle Position, die aktuelle Geschwindigkeit, das aktuelle Drehmoment und das geschätzte, externe Drehmoment der ersten Achse von `ROB_1` werden erfasst.

Argumente

```
GetJointData [\MechUnit] Axis [\Position] [\Speed] [\Torque]
     [\ExtTorque]
```

`[\MechUnit]`

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, für die eine Achse gelesen werden soll. Wenn dieses Argument weggelassen wird, wird die Achse für den angeschlossenen Roboter gelesen.

`Axis`

Datentyp: `num`

Die Nummer der zu lesenden Achse (1 bis 6).

`[\Position]`

Datentyp: `num`

Die aktuelle Position der angegebenen Achse des Roboters oder der externen Achse auf der Armseite. Der Wert wird bei einer rotierenden Achse in Grad und bei einer linearen Achse in mm angegeben.

Mindestens einer der optionalen Parameter `\Position`, `\Speed`, `\Torque`, oder `\ExtTorque` muss verwendet werden.

`[\Speed]`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.86 GetJoinData – Abrufen der gelenkspezifischen Daten
RobotWare - OS
Fortsetzung

Die aktuelle Geschwindigkeit der angegebenen Achse des Roboters oder der externen Achse auf der Armseite. Der Wert wird bei einer rotierenden Achse in Grad/Sekunden und bei einer linearen Achse in mm/Sekunden angegeben.

Mindestens einer der optionalen Parameter `\Position`, `\Speed`, `\Torque`, oder `\ExtTorque` muss verwendet werden.

`[\Torque]`

Datentyp: `num`

Das aktuelle Drehmoment der angegebenen Achse des Roboters oder der externen Achse auf der Armseite in Nm.

Mindestens einer der optionalen Parameter `\Position`, `\Speed`, `\Torque`, oder `\ExtTorque` muss verwendet werden.

`[\ExtTorque]`

Datentyp: `num`

Das aktuelle, geschätzte externe Drehmoment der angegebenen Achse des Roboters oder der externen Achse auf der Armseite in Nm.

Mindestens einer der optionalen Parameter `\Position`, `\Speed`, `\Torque`, oder `\ExtTorque` muss verwendet werden.

Programmabarbeitung

Die Instruktion erfasst die Position, die Geschwindigkeit, das Drehmoment und das geschätzte, externe Drehmoment des Roboters und der externen Achsen.

Die erfassten Werte können auch mithilfe von *TuneMaster* mit den Prüfsignalnummern 4000, 4001, 4002 und 4003 angezeigt werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_AXIS_PAR`	Parameter "Achse" in der Instruktion ist falsch.
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.

Syntax

```
GetJointData
  ['\' MechUnit ':=' < variable (VAR) of mecunit> ',']
  [Axis ':=' ] < expression (IN) of num>
  [ '\' Position ':=' < variable (VAR) of num> ]
  [ '\' Speed ':=' < variable (VAR) of num> ]
  [ '\' Torque ':=' < variable (VAR) of num> ]
  [ '\' ExtTorque ':=' < variable (VAR) of num> ] ';'
```


© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.87 GetSysData - Abrufen von Systemdaten

Verwendung

`GetSysData` ruft den Wert und den optionalen Symbolnamen für die aktuellen Systemdaten des angegebenen Typs ab.

Mit dieser Instruktion können Sie Daten und Namen des gegenwärtig aktiven Werkzeugs, des Werkobjekts sowie die Nutzlast oder die Gesamtlast des Roboters in der aktuellen oder verbundenen oder in einer beliebigen benannten Bewegungstask abrufen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `GetSysData`:

Beispiel 1:

```
PERS tooldata curtoolvalue := [TRUE, [[0, 0, 0], [1, 0, 0, 0]],
     [2, [0, 0, 2], [1, 0, 0, 0], 0, 0, 0]];
VAR string curtoolname;
GetSysData curtoolvalue;
```

Kopiert den aktuell aktiven Werkzeugdatenwert in die persistente Variable `curtoolvalue`.

Beispiel 2:

```
GetSysData curtoolvalue \ObjectName := curtoolname;
```

Kopiert auch den aktuellen aktiven Werkzeugnamen in die Variable `curtoolname`.

Beispiel 3

```
PERS loaddata curload;
PERS loaddata piece:=[2.8,[-38.2,-10.1,-73.6],[1,0,0,0],0,0,0];
PERS loaddata
     tool2piece:=[13.1,[104.5,13.5,115.9],[1,0,0,0],0,0,0.143];
PERS tooldata tool2 := [TRUE, [[138.695,150.023,98.9783],
     [0.709396,-0.704707,-0.00856676,0.00851007]],
     [10,[105.2,-3.8,118.7], [1,0,0,0],0,0,0.123]];
VAR string name;
..
IF GetModalPayloadMode() = 1 THEN
  GripLoad piece;
  MoveL p3, v1000, fine, tool2;
  ..
  ..
  ! Get current payload
  GetSysData curload \ObjectName := name;
ELSE
  MoveL p30, v1000, fine, tool2\TLoad:=tool2piece;
  ..
  ..
  ! Get current total load
  GetSysData curload \ObjectName := name;
ENDIF
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.87 GetSysData - Abrufen von Systemdaten

RobotWare - OS

Fortsetzung

Kopiert die gegenwärtig aktive Nutzlast und den Namen in die Variable `name`, wenn `ModalPayLoadMode` 1 ist.

Kopiert die gegenwärtig aktive Gesamtlast und den Namen in die Variable `name`, wenn `ModalPayLoadMode` 0 ist.

Argumente

```
GetSysData [\TaskRef]|[\TaskName] DestObject[\ObjectName ]
```

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, von der die Daten der aktuellen aktiven Systemdaten gelesen werden sollten.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“ (die Variablenidentität für die Task `T_ROB1` lautet z. B. `T_ROB1Id`).

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, von der die aktuellen aktiven Systemdaten gelesen werden sollten.

Wenn keines der Argumente `\TaskRef` oder `\TaskName` angegeben ist, wird die aktuelle Task verwendet.

`DestObject`

Datentyp: `anytype`

Persistente Variable zur Speicherung des aktuell aktiven Systemdatenwerts.

Der Datentyp dieses Arguments bestimmt auch die Art der abzurufenden Systemdaten (Werkzeug, Werkobjekt oder Nutzlast/Gesamtlast). Bei der Verwendung des optionalen Arguments TLoad für Bewegungsinstruktionen wird die Gesamtlast anstelle der Nutzlast abgerufen, wenn der Datentyp loaddata verwendet wird.

Datentyp	Typ der Systemdaten
tooldata	Werkzeug
wobjdata	Werkobjekt
loaddata	Nutzlast/Gesamtlast

Datenfeld- oder Datensatzkomponente kann nicht verwendet werden.

`[\ObjectName]`

Datentyp:`string`

Optionales Argument (Variable oder Persistente), um auch den aktuell aktiven Systemdatennamen abzurufen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Beim Ausführen der Instruktion `GetSysData` wird der aktuelle Datenwert in der im Argument `DestObject` angegebenen persistenten Variable gespeichert.

Wenn das Argument `\ObjectName` verwendet wird, wird der Name der aktuellen Daten in der im Argument `ObjectName` angegebenen Variablen oder Persistenten gespeichert.

Aktuelle Systemdaten für das Werkzeug, das Werkobjekt oder die Gesamtlast werden durch Abarbeitung einer beliebigen Bewegungsinstruktion aktiviert. Die Nutzlast wird durch die Abarbeitung der Instruktion `GripLoad` aktiviert.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_NOT_MOVETASK`	Die Argumente `\TaskRef` oder `\TaskName` geben eine Nicht-Bewegungstask an. **Hinweis** Es wird kein Fehler erzeugt, wenn die Argumente `\TaskRef` oder `\TaskName` die Task ohne Bewegung angeben, die diese Funktion ausführen `GetSysData` (Referenz zur eigenen Task ohne Bewegung). Die aktuellen Systemdaten werden dann von der verbundenen Bewegungstask geholt.
`ERR_TASKNAME`	Wenn der Name der Programmtask im Argument `\TaskName` im System nicht gefunden werden kann, wird die Systemvariable `ERRNO` auf `ERR_TASKNAME` gesetzt.

Syntax

```
GetSysData
  ['\' TaskRef' :=' <variable (VAR) of taskid>]
  |['\' TaskName' :=' <expression (IN) of string>]
  [ DestObject' :='] <persistent(PERS) of anytype>
  ['\'ObjectName' :=' <variable or persistent (INOUT) of string>]';'
```

Weitere Informationen

Information über	Siehe
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition einer Nutzlast	*loaddata - Lastdaten auf Seite 1723*
Einstellen von Systemdaten	*SetSysData - Setzen von Systemdaten auf Seite 720*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.88 GetTrapData - Abrufen von Interrupt-Daten für aktuellen Interrupt

Verwendung

`GetTrapData` bezieht in einer Interrupt-Routine alle Informationen über den Interrupt, der die Abarbeitung der Interrupt-Routine verursacht hat.

Wird in Interrupt-Routinen verwendet, die durch die Instruktion `IError` vor Verwendung der Instruktion `ReadErrData` erzeugt wurden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `GetTrapData`:

Siehe auch *Weitere Beispiele auf Seite 238*.

Beispiel 1:

```
VAR trapdata err_data;
GetTrapData err_data;
```

Speichert Interrupt-Informationen in der Non-Value-Variablen `err_data`.

Argumente

```
GetTrapData TrapEvent
```

`TrapEvent`

Datentyp: `trapdata`

Variable zur Speicherung der Informationen über die Ursache des Interrupts.

Weitere Beispiele

Weitere Beispiele für die Instruktion `GetTrapData` sind unten aufgeführt.

Beispiel 1:

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Wenn ein Fehler in der Interrupt-Routine `trap_err` festgehalten wird, werden Fehlerdomäne, Fehlernummer und Fehlertyp in geeigneten Non-Value-Variablen des Typs `trapdata` gespeichert.

Einschränkung

Diese Instruktion kann nur in einer `TRAP`-Routine verwendet werden.

Syntax

```
GetTrapData
  [TrapEvent ':='] <variable (VAR) of trapdata>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt-Daten für aktuellen TRAP	*trapdata – Interrupt-Daten für aktuellen TRAP auf Seite 1831*
Anfordern eines Interrupts bei Fehlern	*IError - Anfordern eines Interrupts bei Fehlern auf Seite 255*
Abrufen von Informationen über einen Fehler	*ReadErrData - Abrufen von Informationen über einen Fehler auf Seite 590*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.89 GOTO - Sprung zu einer anderen Instruktion

Verwendung

`GOTO` wird verwendet, um die Programmabarbeitung in eine andere Zeile (ein Label) innerhalb derselben Routine zu verlegen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `GOTO`:

Beispiel 1:

```
GOTO next;
...
next:
```

Die Programmabarbeitung fährt mit der Instruktion fort, die auf "next" folgt.

Beispiel 2:

```
reg1 := 1;
next:
...
reg1 := reg1 + 1;
IF reg1<=5 GOTO next;
```

Die Abarbeitung wird viermal an `next` übertragen (für `reg1`= 2, 3, 4, 5).

Beispiel 3

```
IF reg1>100 THEN
  GOTO highvalue
ELSE
  GOTO lowvalue
ENDIF
lowvalue:
...
GOTO ready;
highvalue:
...
ready:
```

Wenn `reg1` größer als `100` ist, wird die Abarbeitung an das Label `highvalue` übertragen. Andernfalls wird die Abarbeitung an das Label `lowvalue` übertragen.

Argumente

```
GOTO Label
```

`Label`

Identifier

Das Label, ab dem die Programmabarbeitung fortfahren soll.

Einschränkungen

Die Programmabarbeitung kann nur an ein Label innerhalb derselben Routine versetzt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Es ist nur möglich, die Programmabarbeitung an ein Label in einer `IF`- oder `TEST`-Instruktion zu übertragen, wenn sich die `GOTO` -Instruktion in demselben Zweig dieser Instruktion befindet.

Es ist nur möglich, die Programmabarbeitung an ein Label in einer `FOR`- oder `WHILE`-Instruktion zu übertragen, wenn sich die `GOTO`-Instruktion ebenfalls in dieser Instruktion befindet.

Syntax

```
GOTO <identifier>';'
```

Weitere Informationen

Information über	Siehe
Schild	*Label - Zeilenname auf Seite 345*
Andere Instruktionen, die den Programmablauf ändern.	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.90 GripLoad - Definieren der Nutzlast des Roboters

Verwendung

`GripLoad` wird zur Definition der Nutzlast verwendet, die der Roboter in seinem Greifer hält.

Beschreibung

`GripLoad` gibt an, welche Last der Roboter trägt. Die angegebene Last wird durch das Steuersystem verwendet, damit die Roboterbewegungen ideal gesteuert werden können.

Die Nutzlast wird mit der Instruktion `GripLoad` aufgenommen/abgelegt, die das Gewicht der Nutzlast dem Gewicht des Greifers hinzufügt oder von ihm abzieht.

WARNUNG

Die korrekte aktuelle Werkzeuglast sowie, falls vorhanden, die Nutzlast des Roboters müssen unbedingt definiert werden. Falsche Definitionen der Lastdaten können zu Überlastung der mechanischen Roboterstruktur führen. Es besteht zudem ein Risiko, dass die Geschwindigkeit im manuell reduzierten Geschwindigkeitsmodus überschritten werden kann.

Die Angabe falscher Lastdaten kann folgende Konsequenzen haben:

- Der Roboter verwendet möglicherweise nicht seine maximale Kapazität.
- Die Bahngenauigkeit wird beeinträchtigt und es besteht das Risiko des Überschwingens.
- Die mechanische Struktur kann überlastet werden.

Die Steuerung überwacht ununterbrochen die Last und schreibt ein Ereignisprotokoll, wenn die Last höher als erwartet ist. Dieses Ereignisprotokoll wird gespeichert und Steuerungsspeicher protokolliert.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `GripLoad`.

Beispiel 1:

```
Set doGripper;
!wait to grip
WaitTime 0.3;
GripLoad piece1;
```

Aufnehmen der Nutzlast `piece1` wird gleichzeitig mit dem Greifen der Last definiert.

Beispiel 2:

```
Reset doGripper;
!wait to release
WaitTime 0.3;
GripLoad load0;
```

Ablegen der Nutzlast piece1 wird gleichzeitig mit dem Freigeben der Nutzlast piece1 definiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
GripLoad Load
```

`Load`

Datentyp: `loaddata`

Die Lastdaten, die die aktuelle Nutzlast beschreiben.

Das Programm kann ohne Nutzlast getestet werden, indem ein digitales Eingangssignal verwendet wird, das mit der Systemeingabe `SimMode` (Simulationsmodus) verbunden ist. Wenn das digitale Eingangssignal auf 1 gesetzt ist, wird `loaddata` in der `GripLoad`-Instruktion nicht berücksichtigt, und es wird nur `loaddata` in den aktuellen `tooldata` verwendet.

Programmabarbeitung

Die angegebene Last gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue `GripLoad`-Instruktion ausgeführt wird.

Die angegebene Last beeinflusst die Leistung des Roboters.

Die Standardlast (`load0`) von 0 kg wird automatisch eingestellt

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Die Nutzlast wird für die mechanische Einheit aktualisiert, die von der aktuellen Programmtask gesteuert wird. Wenn `GripLoad` von einer Nicht-Bewegungstask ausgeführt wird, wird die Nutzlast für die mechanische Einheit aktualisiert, die von der verbundenen Bewegungstask gesteuert wird.

Syntax

```
GripLoad
  [Load ':='] <persistent (PERS) of loaddata>';'
```

Weitere Informationen

Information über	Siehe
Lastidentifikation der Werkzeuglast, Nutzlast oder Armlast	*Bedienungsanleitung - IRC5 mit FlexPendant*, Abschnitt *Programming and testing - Service routines*
Definieren der Nutzlast für mechanische Einheiten	*MechUnitLoad - Definieren einer Nutzlast für eine mechanische Einheit auf Seite 372*
Definition der Lastdaten	*loaddata - Lastdaten auf Seite 1723*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.91 HollowWristReset - Hohlgelenk zurücksetzen

Verwendung

`HollowWristReset` (*Reset hollow wrist*) setzt die Position von Handgelenkachsen an Manipulatoren mit Hohlgelenken (wie z. B. IRB 5402 und IRB 5403) zurück.

Die Instruktion macht ein Zurücksetzen der Handgelenkachsen 4 und 5 überflüssig, nachdem sie eine oder mehrere Umdrehungen gedreht wurden. Nach dem Ausführen der Instruktion `HollowWristReset` kann es sein, dass die Handgelenkachsen weiterhin in dieselbe Richtung gedreht werden.

Beschreibung

Durch `HollowWristReset` wird die Erstellung von Anwendungsprogrammen vereinfacht. Sie müssen beim Zeitpunkt der Programmierung nicht sicherstellen, dass die Handgelenkposition innerhalb von ±2 Umdrehungen liegt; so wird eventuell die Zyklusdauer verkürzt, da der Roboter das Handgelenk nicht zurückdrehen muss. Es gibt eine Einschränkung von ±144 Umdrehungen für das Drehen der Gelenke 4 und 5, bevor die Handgelenkposition mithilfe von `HollowWristReset` zurückgesetzt werden muss. Der Programmierer des Roboters muss diese Einschränkung beachten und bei der Planung der Roboterprogramme berücksichtigen. Um sicherzustellen, dass die Beschränkung auf 144 Umdrehungen nach der mehrmaligen Durchführung von Programmen zum *Drehen des Handgelenks* nicht überschritten wird, sollte der Roboter in jedem Programm (oder Zyklus/Routine/Modul usw.) komplett angehalten und in die absolute Position zurückgesetzt werden. Beachten Sie, dass sich bei der Abarbeitung der Instruktion `HollowWristReset` keine Achse bewegen darf. Solange diese Einschränkungen beachtet werden, können sich die Gelenke 4 und 5 bei der Programmabarbeitung dauernd und unabhängig von Gelenk 6 drehen.

Verwenden Sie `HollowWristReset` anstelle von `IndReset`, um das Hohlgelenk zurückzusetzen, da diese Instruktion die Einschränkungen für Gelenk 6 berücksichtigt und dadurch ein zu starkes Verdrehen der Farbschläuche/Kabel vermieden wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `HollowWristReset`:

Beispiel 1:

```
MoveL p10,v800,fine,paintgun1\WObj:=workobject1;
HollowWristReset;
```

Alle aktiven Achsen werden durch einen Stopppunkt angehalten und das Handgelenk wird zurückgesetzt.

Einschränkungen

Bei der Ausführung der Instruktion `HollowWristReset` müssen alle aktiven Achsen angehalten sein.

Die Handgelenken müssen zurückgesetzt werden, bevor ein Handgelenk die Umdrehungsgrenze von ±144 (d. h. 51840 Grad/ 904 Radiant) erreicht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Bei jedem Programmstopp, Not-Halt, Stromausfall usw. speichert die Steuerung den Bahnkontext, um die Bahn an dem Punkt, an dem die Unterbrechung stattfand, wieder aufzunehmen und den Roboter die Abarbeitung des Programms fortsetzen lassen zu können. Wenn der Manipulator im Einrichtbetrieb zwischen einem Stopp und einem Neustart aus der Bahn bewegt wurde, wird der Bediener durch folgende Meldung auf dem FlexPendant davon in Kenntnis gesetzt: **Not on path! Robot has been moved after program stop. Should the robot return to the path on Start? Yes**/**No**/**Cancel** (Außerhalb der Bahn! Roboter wurde nach dem Programmstopp bewegt. Soll der Roboter beim Start die Bahn wieder aufnehmen? Ja/Nein/Abbrechen) Auf diese Weise kann der Roboter vor einem Neustart wieder in seine Bahn zurückkehren. Im Automatikmodus kehrt der Roboter automatisch auf seine Bahn zurück.

`HollowWristReset` entfernt den Bahnkontext. Das bedeutet, dass es im Fall eines Programmneustarts nicht möglich ist, in die Bahn zurückzukehren, wenn in der Zwischenzeit die Instruktion `HollowWristReset` ausgeführt wurde. Wenn diese Instruktion manuell ausgeführt wird (Serviceroutine), sollte sie nur dann ausgeführt werden, wenn der Roboter nicht in seine Bahn zurückkehren muss. Das heißt also nach der kompletten Abarbeitung eines Programms oder der schrittweisen Abarbeitung einer Instruktion, wenn der Manipulator dabei nicht manuell aus der Bahn bewegt wird usw.

Syntax

```
HollowWristReset ';'
```

Weitere Informationen

Information über	Siehe
Verwandte Systemparameter	*Technisches Referenzhandbuch - Systemparameter*
Zurück zur Bahn	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.92 ICap - verbindet CAP-Ereignisse mit Trap-Routinen

Verwendung

`ICap` wird verwendet, um eine Interrupt-Nummer (die bereits mit einer Trap-Routine verbunden ist) mit einem bestimmten CAP-Ereignis zu verbinden. In den Argumenten unten finden Sie eine Liste der verfügbaren Ereignisse. Beim Benutzen von `ICap` wird eine Zuordnung zwischen einem spezifischen Prozessereignis und einer benutzerdefinierten Trap-Routine erzeugt. Mit anderen Worten: die betreffende Trap-Routine wird ausgeführt, wenn das zugehörige CAP-Ereignis eintritt.

Wir empfehlen, die Traps in einer Hintergrundtask zu positionieren.

Grundlegendes Beispiel

Unten ist ein Beispiel, bei dem das CAP-Ereignis CAP_START mit der Trap-Routine `start_trap` verbunden ist.

```
VAR intnum start_intno:=0;
...

TRAP start_trap
  ! This routine will be executed when the event CAP_START is
       reported from the core
  ! Do what you want to do
ENDTRAP

PROC main()
  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
  CapL p1, v100, cdata, weavestart, weave, z50, gun1;
ENDPROC
```

Argumente

```
ICap Interrupt Event
```

Interrupt

Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Event

Datentyp: `num`

Die CAP-Ereignisnummer muss dem Interrupt zugeordnet werden. Diese Ereignisse sind vordefinierte Konstanten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Verfügbare CAP-Ereignisse

Siehe Abschnitt *Coupling between phases and events* in *Application manual - Continuous Application Platform*.

Ereignisse	Phase	Ereignisnummer	Beschreibung
CAP_START		0	Dieses Ereignis tritt auf, sobald der CAP-Prozess gestartet wird.
START_PRE	PRE	1	Dieses Ereignis tritt ein, wenn die Überwachung der PRE-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
PRE_STARTED	PRE	2	Dieses Ereignis tritt auf, wenn alle Anforderungen der PRE Überwachungsliste erfüllt werden, das heißt, wenn die PRE_START-Phase gestartet wurde. Wenn ein *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
START_MAIN	START	3	Dieses Ereignis tritt auf, wenn die PRE_START-Phase beendet wurde und die MAIN-Phase gestartet wird.
MAIN_STARTED	START	4	Dieses Ereignis tritt auf, wenn alle Bedingungen der START Überwachungsliste erfüllt wurden, das heißt, wenn die MAIN-Phase gestartet wird.
STOP_WEAVESTART	MAIN	5	Dieses Ereignis tritt vor jedem Pendelstart auf, jedoch nur, wenn der Pendelstart angeordnet wird. Wenn ein *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
WEAVESTART_REGAIN	MAIN	6	Dieses Ereignis tritt ein, wenn der Roboter nach einem Pendelstart wieder auf die Bahn zurückgekehrt ist. Bei einem *flying start* wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
MOTION_DELAY	MAIN	7	Dieses Ereignis tritt nach der Verzögerung bei Bewegungsstart ein, soweit vorhanden. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
STARTSPEED_TIME	MAIN	8	Dieses Ereignis tritt ein, wenn die Zeit für die Verwendung von *Start Speed* abläuft und es Zeit ist, zu den Hauptbewegungsdaten zu wechseln.
MAIN_MOTION	MAIN	9	Dieses Ereignis tritt ein, wenn die Hauptbewegung mit Prozess aktiviert ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.92 ICap - verbindet CAP-Ereignisse mit Trap-Routinen

Continuous Application Platform (CAP)

Fortsetzung

Ereignisse	Phase	Ereignisnummer	Beschreibung
MOVE_STARTED	MAIN	10	Dieses Ereignis tritt auf, sobald der Roboter sich entlang der Prozessbahn zu bewegen beginnt. Wenn ein *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
RESTART	MAIN	11	Dieses Ereignis tritt ein, wenn Neustart angeordnet wird.
NEW_INSTR	MAIN	12	Dieses Ereignis tritt auf, wenn eine neue `CapL`- oder `CapC`-Instruktion aus dem RAPID-Programm abgerufen wird.
AT_POINT	MAIN	13	Dieses Ereignis tritt bei jedem robtarget auf der Prozessbahn auf, mit Ausnahme des Start- und Zielpunkts.
AT_RESTARTPOINT	MAIN	14	Dieses Ereignis tritt ein, wenn der Roboter den Neustart-Abstand nach einem Stopp auf der Bahn zurückläuft.
LAST_SEGMENT	MAIN	15	Dieses Ereignis tritt am Startpunkt des letzten Segments auf.
PROCESS_END_POINT	MAIN	16	Dieses Ereignis tritt auf, wenn der Roboter den Endpunkt des Prozesses erreicht, das heißt, wo der Prozess beendet werden soll. Bei einem *flying end* wird kein Ereignis verteilt.
END_MAIN	END_MAIN	17	Dieses Ereignis tritt an dem Punkt auf der Prozessbahn ein, wo die Überwachung der Endsequenz gestartet wird, das heißt, wenn der Roboter den Endpunkt des Prozesses erreicht.
MAIN_ENDED	END_MAIN	18	Dieses Ereignis tritt auf, wenn alle Bedingungen der END_MAIN Überwachungsliste erfüllt wurden, das heißt, wenn der Prozess als beendet betrachtet wird.
PATH_END_POINT		19	Dieses Ereignis tritt auf, wenn der Roboter den Endpunkt der Bahn erreicht, d.h. am Feinpunkt oder in der Mitte der Zone der letzten CAP-Instruktion (bei *flying end*).
PROCESS_ENDED		20	Dieses Ereignis tritt nur ein, wenn beide Prozesse am Feinpunkt oder in der Mitte der Zone der letzten CAP-Instruktion beendet werden (bei *flying end*).
END_POST1	END_POST1	21	Dieses Ereignis tritt auf, wenn es Zeit ist, die POST1-Phase zu beenden, das heißt, wenn es Zeit ist, von der POST1- zur POST2-Phase zu wechseln. Bei einem *flying end* wird kein Ereignis verteilt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Ereignisse	Phase	Ereignisnummer	Beschreibung
POST1_ENDED	END_POST1	22	Dieses Ereignis tritt auf, wenn alle Bedingungen der END_POST1 Überwachungsliste erfüllt sind, das heißt, wenn die POST1-Phase erfolgreich beendet und die POST2-Phase gestartet wurde. Bei einem *flying end* wird kein Ereignis verteilt.
END_POST2	END_POST2	23	Dieses Ereignis tritt auf, wenn die POST2-Phase zu Ende ist, das heißt, wenn es Zeit ist, den Prozess endgültig abzuschließen. Bei einem *flying end* wird kein Ereignis verteilt.
POST2_ENDED	END_POST2	24	Dieses Ereignis tritt auf, wenn alle Bedingungen der END_POST2 Überwachungsliste erfüllt sind, das heißt, wenn die POST2-Phase und damit der ganze Prozess erfolgreich beendet wurde. Bei einem *flying end* wird kein Ereignis verteilt.
CAP_STOP		25	Dieses Ereignis ist ein Pflichtereignis. Wenn ein anderes Ereignis verwendet wird, muss dieses Ereignis ebenfalls definiert werden. Die Ereignis-/Trap-Routine wird so schnell wie möglich ausgeführt, nachdem die Steuerung aufgrund eines Fehlers oder eines Programms angehalten wurde. Ein Fehler kann ein behebbarer Fehler in CAP, ein schwerwiegender Fehler in CAP oder ein interner Fehler sein, der die Steuerung stoppt. Der Code, der für diese Trap abgearbeitet wird, muss alle externen Geräte in einen sicheren Zustand bringen, d.h. zum Beispiel alle externen E/A-Signale zurücksetzen.
CAP_PF_RESTART	MAIN	26	Dieses Ereignis tritt ein, wenn Neustart angeordnet wird.
EQUIDIST	MAIN	27	Dieses Ereignis wird gesendet, wenn es durch Instruktion`CapEquiDist` angeordnet wird.
AT_ERRORPOINT	MAIN	28	Dieses Ereignis tritt nach dem Neustart auf, wenn der TCP die Position des Überwachungsfehlers erreicht.
FLY_START	MAIN	29	Dieses Ereignis tritt bei der Verwendung von *flying start* auf. Diese Ereignis ist nur bei einem *flying start* verfügbar.
FLY_END	MAIN	30	Dieses Ereignis tritt bei der Verwendung von *flying end* auf. Diese Ereignis ist nur bei einem *flying end* verfügbar.
LAST_INSTR_ENDED	MAIN	31	Dieses Ereignis tritt ein, wenn eine RAPID-Abarbeitung der letzten CAP-Instruktion während eines *flying end* beendet wurde. Dieses Ereignis ist nur bei einem *flying end* verfügbar.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.92 ICap - verbindet CAP-Ereignisse mit Trap-Routinen
Continuous Application Platform (CAP)
Fortsetzung

Ereignisse	Phase	Ereignisnummer	Beschreibung
END_PRE	PRE	32	Dieses Ereignis tritt ein, wenn die Überwachung der PRE-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
PRE_ENDED	PRE	33	Dieses Ereignis tritt ein, wenn die Überwachung der PRE-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
START_POST1	POST1	34	Dieses Ereignis tritt ein, wenn die Überwachung der POST1-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
POST1_STARTED	POST1	35	Dieses Ereignis tritt ein, wenn die Überwachung der POST1-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
START_POST2	POST2	36	Dieses Ereignis tritt ein, wenn die Überwachung der POST1-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.
POST2_STARTED	POST2	37	Dieses Ereignis tritt ein, wenn die Überwachung der POST1-Phase, falls vorhanden, aktiviert wird. Wenn *flying start* verwendet wird, wird kein Ereignis verteilt, weil es bereits eine TCP-Bewegung gibt. Bei einem Neustart wird dieses Ereignis verteilt.

Einschränkungen

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
PROC setup_events ()
  VAR intnum start_intno;
  IDelete start_intno;
  CONNECT start_intno WITH start_trap;
  ICap start_intno, CAP_START;
ENDPROC
```

Die gesamte Aktivierung der Interrupts erfolgt am Anfang des Programms. Diese Instruktionen werden dann außerhalb des Hauptablaufs des Programms beibehalten. Die `ICap`-Instruktion wird nur einmal ausgeführt, z.B. von der

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Ereignisroutine des Systemstarts. Es empfiehlt sich, die Traps in einer Hintergrundtask zu platzieren.

Syntax

```
ICap
  [Interrupt ':='] < variable (IN) of intnum > ','
  [Event ':='] < variable (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Interrupt mit einer Trap verbinden	*CONNECT - Verbinden eines Interrupts mit einer Interrupt-Routine auf Seite 161*
Ein mit einer Trap verbundenes Interrupt abbrechen	*IDelete - Abbrechen eines Interrupts auf Seite 252*
Datentyp `intnum`	*intnum - Interrupt-Identität auf Seite 1716*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.93 IDelete - Abbrechen eines Interrupts

Verwendung

`IDelete` (*Interrupt Delete*) wird zum Abbrechen (Löschen) einer Interrupt-Einplanung verwendet.

Wenn der Interrupt nur temporär blockiert werden soll, verwenden Sie die Instruktion `ISleep` oder `IDisable`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IDelete`:

Beispiel 1:

```
IDelete feeder_low;
```

Der Interrupt `feeder_low` wird abgebrochen.

Argumente

```
IDelete Interrupt
```

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität.

Programmabarbeitung

Die Definition des Interrupts wird vollständig gelöscht. Um ihn erneut zu definieren, muss er zuerst wieder mit der Interrupt-Routine verbunden werden.

Es wird empfohlen, der Instruktion `IDelete` einen Stopppunkt voranzustellen. Anderenfalls wird der Interrupt deaktiviert, bevor der Endpunkt der Bewegungsbahn erreicht wird.

Interrupts müssen nicht gelöscht werden; dies erfolgt automatisch:

- beim Laden eines neuen Programms,
- wenn das Programm vom Anfang gestartet wird
- wenn der Programmzeiger an den Beginn einer Routine versetzt wird.

Syntax

```
IDelete [ Interrupt ':=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Temporäres Blockieren eines Interrupts	*ISleep - Deaktivieren eines Interrupts auf Seite 334*
Temporäres Blockieren aller Interrupts	*IDisable - Blockieren von Interrupts auf Seite 253*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.94 IDisable - Blockieren von Interrupts

Verwendung

`IDisable`*(Interruptdeaktivierung)* wird verwendet, um alle Interrupts zeitweilig zu deaktivieren. Die Instruktion kann z. B. in einem besonders empfindlichen Programmteil verwendet werden, in dem Interrupts den normalen Programmablauf stören würden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IDisable`:

Beispiel 1:

```
IDisable;
FOR i FROM 1 TO 100 DO
  character[i]:=ReadBin(sensor);
ENDFOR
IEnable;
```

Während des Lesens von dem Sensor, sind keine Interrupts zulässig. Nach dem Lesen sind Interrupts wieder zulässig.

Programmabarbeitung

Interrupts, die während einer wirksamen `IDisable`-Instruktion auftreten, werden in eine Warteschlange gestellt. Sobald Interrupts wieder freigegeben sind, werden sie sofort in der Reihenfolge ihres Auftretens in der Warteschlange vom Programm abgearbeitet.

`IEnable` ist standardmäßig aktiviert. `IEnable` wird automatisch festgelegt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass der Abarbeitungsbefehl verloren geht
- nach Abarbeiten eines Zyklus (unter Umgehung von `main`) oder Abarbeiten von `ExitCycle`.

Syntax

```
IDisable';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Freigeben von Interrupts	*IEnable - Freigeben von Interrupts auf Seite 254*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.95 IEnable - Freigeben von Interrupts

Verwendung

`IEnable`(*Interrupt Enable*) wird zum Freigeben von Interrupts bei der Programmabarbeitung verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IEnable`:

Beispiel 1:

```
IDisable;
FOR i FROM 1 TO 100 DO
  character[i]:=ReadBin(sensor);
ENDFOR
IEnable;
```

Während des Lesens von dem Sensor, sind keine Interrupts zulässig. Nach dem Lesen sind Interrupts wieder zulässig.

Programmabarbeitung

Interrupts, die während einer wirksamen `IDisable`-Instruktion auftreten, werden in eine Warteschlange gestellt. Sobald Interrupts wieder freigegeben sind (`IEnable`), werden sie sofort in der Reihenfolge ihres Auftretens in der Warteschlange vom Programm abgearbeitet. Die Programmabarbeitung fährt dann im normalen Programm fort und später auftretende Interrupts werden wie üblich behandelt.

Interrupts sind immer erlaubt, wenn das Programm vom Anfang gestartet wird. Interrupts, die durch die Anweisung `ISleep` blockiert wurden, werden nicht durch die Instruktion `IEnable` beeinflusst.

Syntax

```
IEnable';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Verbieten von Interrupts	*IDisable - Blockieren von Interrupts auf Seite 253*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.96 IError - Anfordern eines Interrupts bei Fehlern

Verwendung

`IError` (*Interrupt Errors*) wird verwendet, um einen Interrupt anzufordern und freizugeben, wenn ein Fehler auftritt.

Fehler, Warnungen oder Statusänderungen können mit `IError` protokolliert werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IError`:

Siehe auch *Weitere Beispiele auf Seite 256*.

Beispiel 1:

```
VAR intnum err_int;
...
PROC main()
  CONNECT err_int WITH err_trap;
  IError COMMON_ERR, TYPE_ALL, err_int;
```

Fordert jedes Mal einen Interrupt in RAPID und Abarbeitung der `TRAP`-Routine `err_trap` an, wenn ein Fehler, eine Warnung oder eine Statusänderung im System generiert wird.

Argumente

```
IError ErrorDomain [\ErrorId] ErrorType Interrupt
```

`ErrorDomain`

Datentyp: `errdomain`

Die zu überwachende Fehlerdomäne. Siehe vordefinierten Datentyp `errdomain`. Zur Angabe einer Domäne verwenden Sie `COMMON_ERR`.

`[ \ErrorId ]`

Datentyp: `num`

Optional die Nummer eines bestimmten Fehlers, der zu überwachen ist. Die Fehlernummer muss ohne die erste Ziffer (Fehlerdomäne) angegeben werden.

Zum Beispiel: **10008 Programm-Neustart** muss als *0008* oder nur als *8* angegeben werden.

`ErrorType`

Datentyp: `errtype`

Typ des zu überwachenden Ereignisses, z. B. Fehler, Warnung oder Statusänderung. Siehe vordefinierten Datentyp `errtype`. Zur Angabe eines Typs verwenden Sie `TYPE_ALL`.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.96 IError - Anfordern eines Interrupts bei Fehlern
RobotWare - OS
Fortsetzung

Programmabarbeitung

Die entsprechende Interrupt-Routine wird automatisch bei Auftreten eines Fehlers in der angegebenen Domäne, mit dem angegebenen Typ und optional mit der angegebenen Fehlernummer aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IError` sind unten aufgeführt.

```
VAR intnum err_interrupt;
VAR trapdata err_data;
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
PROC main()
  CONNECT err_interrupt WITH trap_err;
  IError COMMON_ERR, TYPE_ERR, err_interrupt;
  ...
  IDelete err_interrupt;
  ...
ENDPROC
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
  ! Set domain no 1 ... 11
  SetGO go_err1, err_domain;
  ! Set error no 1 ...9999
  SetGO go_err2, err_number;
ENDTRAP
```

Bei Auftreten eines Fehlers (nur Fehler, nicht Warnung oder Statusänderung) wird die Fehlernummer in der Interrupt-Routine abgerufen und mit ihrem Wert werden zwei Gruppen von digitalen Ausgangssignalen gesetzt.

Einschränkung

Es ist nicht möglich, einen Interrupt auf interne Fehler anzufordern.

In einer Task vom Typ „Normal“ wird das Ereignis bei einem Programmstopp verworfen. Daher können in einer Normal-Task nicht alle Ereignisse abgerufen werden. Um alle Ereignisse abzurufen, muss die Task vom Typ „static“ oder „semi-static“ sein.

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
VAR intnum err_interrupt;
PROC main ( )
  CONNECT err_interrupt WITH err_trap;
  IError COMMON_ERR, TYPE_ERR, err_interupt;
  WHILE TRUE DO
  :
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  :
  ENDWHILE
ENDPROC
```

Interrupts werden am Beginn des Programms aktiviert. Diese Instruktionen werden zu Beginn dann außerhalb des Hauptablaufs des Programms gehalten.

```
VAR intnum err_interrupt;
PROC main ( )
  CONNECT err_interrupt WITH err_trap;
  IError COMMON_ERR, TYPE_ERR, err_interupt;
  :
  :
  IDelete err_interrupt;
ENDPROC
```

Der Interrupt wird am Ende des Programms gelöscht und dann erneut aktiviert. Beachten Sie in diesem Fall, dass der Interrupt für eine kurze Zeit inaktiv ist.

Syntax

```
IError
  [ErrorDomain ':='] <expression (IN) of errdomain>
  ['\'ErrorId':=' <expression (IN) of num>','
  [ErrorType' :='] <expression (IN) of errtype> ','
  [Interrupt' :='] <variable (VAR) of intnum>';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Fehlerdomänen, vordefinierte Konstanten	*errdomain – Fehlerdomäne auf Seite 1692*
Fehlertypen, vordefinierte Konstanten	*errtype – Fehlertyp auf Seite 1703*
Abrufen von Interrupt-Daten für aktuellen Interrupt	*GetTrapData - Abrufen von Interrupt-Daten für aktuellen Interrupt auf Seite 238*
Abrufen von Informationen über einen Fehler	*ReadErrData - Abrufen von Informationen über einen Fehler auf Seite 590*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.97 IF - Wenn eine Bedingung erfüllt ist, dann...; sonst...

Verwendung

`IF` wird verwendet, wenn verschiedene Instruktionen abzuarbeiten sind, je nachdem, ob eine Bedingung erfüllt wird oder nicht.

Grundlegende Beispiele

Grundlegende Beispiele für die Instruktion `IF` sind unten aufgeführt.

Siehe auch *Weitere Beispiele auf Seite 259*.

Beispiel 1:

```
IF reg1 > 5 THEN
  Set do1;
  Set do2;
ENDIF
```

Die Signale `do1` und `do2` werden nur gesetzt, wenn `reg1` größer als `5` ist.

Beispiel 2:

```
IF reg1 > 5 THEN
  Set do1;
  Set do2;
ELSE
  Reset do1;
  Reset do2;
ENDIF
```

Die Signale `do1` und `do2` werden abhängig davon gesetzt oder zurückgesetzt, ob `reg1` größer als `5` ist oder nicht.

Argumente

```
IF Condition THEN ...
  {ELSEIF Condition THEN ...}
[ELSE ...]
ENDIF
```

`Condition`

Datentyp: `bool`

Die Bedingung muss erfüllt sein, bevor die Instruktionen zwischen `THEN` und `ELSE/ELSEIF` abgearbeitet werden.

Programmabarbeitung

Die Bedingungen werden in sequenzieller Reihenfolge getestet, bis eine von ihnen erfüllt ist. Die Programmabarbeitung fährt mit den Instruktionen fort, die mit dieser Bedingung verbunden sind. Wenn keine der Bedingungen erfüllt wird, fährt die Programmabarbeitung mit den Instruktionen fort, die auf `ELSE` folgen. Wenn mehrere Bedingungen erfüllt sind, werden nur die Instruktionen abgearbeitet, die mit der ersten erfüllten Bedingung verbunden sind.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `IF` sind unten aufgeführt.

Beispiel 1:

```
IF counter > 100 THEN
  counter := 100;
ELSEIF counter < 0 THEN
  counter := 0;
ELSE
  counter := counter + 1;
ENDIF
```

`counter` wird um `1` inkrementiert. Wenn jedoch der Wert von `counter` außerhalb der Grenze `0-100` liegt, wird `counter` der entsprechende Grenzwert zugewiesen.

Syntax

```
IF <conditional expression> THEN
  <statement list>
{ ELSEIF <conditional expression> THEN
  <statement list> | <EIT> }
[ ELSE
  <statement list> ]
ENDIF
```

Weitere Informationen

Information über	Siehe
Bedingungen (logische Ausdrücke)	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.98 Incr - Inkrementieren um 1

Verwendung

`Incr` wird verwendet, um 1 zu einer numerischen Variablen oder Persistenten zu addieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Incr`:

Siehe auch *Weitere Beispiele auf Seite 260*.

Beispiel 1:

```
Incr reg1;
```

1 wird zu `reg1` addiert, d. h. `reg1:=reg1+1`.

Argumente

```
Incr Name | Dname
```

`Name`

Datentyp: `num`

Der Name der zu ändernden Variablen oder Persistenten.

`Dname`

Datentyp: `dnum`

Der Name der zu ändernden Variablen oder Persistenten.

Weitere Beispiele

Weitere Beispiele für die Instruktion `Incr` sind unten aufgeführt.

Beispiel 1:

```
VAR num no_of_parts:=0;
...
WHILE stop_production=0 DO
  produce_part;
  Incr no_of_parts;
  TPWrite "No of produced parts= "\Num:=no_of_parts;
ENDWHILE
```

Die Anzahl der produzierten Teile wird bei jedem Zyklus am FlexPendant aktualisiert. Die Produktion fährt fort, solange das Eingangssignal `stop_production` nicht gesetzt ist.

Beispiel 2:

```
VAR dnum no_of_parts:=0;
...
WHILE stop_production=0 DO
  produce_part;
  Incr no_of_parts;
  TPWrite "No of produced parts= "\Dnum:=no_of_parts;
ENDWHILE
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Anzahl der produzierten Teile wird bei jedem Zyklus am FlexPendant aktualisiert. Die Produktion fährt fort, solange das Eingangssignal `stop_production` nicht gesetzt ist.

Syntax

```
Incr
  [ Name ':=' ] < var or pers (INOUT) of num >
  | [ Dname' :=' ] < var or pers (INOUT) of dnum >' ;'
```

Weitere Informationen

Information über	Siehe
Dekrementieren der Variable um 1	*Decr - Dekrementieren um 1 auf Seite 189*
Addieren eines Werts zu einer Variablen	*Add – Addiert einen numerischen Wert auf Seite 32*
Ändern von Daten über einen beliebigen Ausdruck, z. B. Multiplikation	*":=" - Zuweisen eines Werts auf Seite 42*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.99 IndAMove - Unabhängige Bewegung in absolute Position

Verwendung

`IndAMove`(*Independent Absolute Movement*) wird verwendet, um den Modus einer Achse in "unabhängig" zu ändern und die Achse in eine bestimmte Position zu bringen.

Eine unabhängige Achse bewegt sich unabhängig von anderen Achsen im Robotersystem. Da die Programmabarbeitung sofort fortgesetzt wird, können andere Instruktionen (einschließlich Positionierungsinstruktionen) ausgeführt werden, während sich die unabhängige Achse bewegt.

Wenn die Achse während einer Umdrehung bewegt werden soll, ist es besser, die Instruktion `IndRMove` zu verwenden. Wenn die Bewegung nur ein geringes Stück von der aktuellen Position entfernt erfolgen soll, muss die Instruktion `IndDMove` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IndAMove`:

Siehe auch *Weitere Beispiele auf Seite 264*.

Beispiel 1:

```
IndAMove Station_A,2\ToAbsPos:=p4,20;
```

Achse `2` von `Station_A` wird mit einer Geschwindigkeit von `20` Grad/s in Position `p4` gebracht.

Argumente

```
IndAMove MecUnit Axis [\ToAbsPos] | [\ToAbsNum] Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1-6)

`[\ToAbsPos]`

To Absolute Position

Datentyp: `robtarget`

Die Achsenposition wird unter `robtarget` angegeben. Es wird nur die Komponente für diese bestimmte `Axis` verwendet. Der Wert wird als Wert für die absolute Position in Grad verwendet (mm für lineare Achsen).

Es hat Auswirkungen auf die Achsenposition, wenn die Position der Achse mithilfe der Instruktion `EOffsSet` oder `EOffsOn` geändert wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Für Roboterachsen muss stattdessen das Argument `\ToAbsNum` verwendet werden.

`[\ToAbsNum]`

To Absolute Numeric value

Datentyp: `num`

Die Achsenposition ist in Grad definiert (in mm für lineare Achsen).

Bei Verwendung dieses Arguments haben Positionsänderungen mithilfe von `EOffsSet` oder `PDispOn` KEINE Auswirkungen auf die Position.

Diese Funktion entspricht derjenigen von `\ToAbsPos`, mit dem Unterschied, dass die Position als numerischer Wert definiert ist, wodurch eine manuelle Änderung der Position problemlos möglich ist.

`Speed`

Datentyp: `num`

Die Achsendrehzahl wird in Grad/s angegeben (in mm/s für lineare Achsen).

`[\Ramp]`

Datentyp: `num`

Verringern der Beschleunigung und der Verzögerung von maximaler Leistung (1 - 100 %, 100 % = maximale Leistung).

Programmabarbeitung

Wenn `IndAMove` ausgeführt wird, bewegt sich die angegebene Achse mit der programmierten Geschwindigkeit zu der angegebenen Achsenposition. Falls `\Ramp` programmiert wurde, wird auch die Beschleunigung/Verzögerung verringert.

Für die Rückkehr der Achse in den Normalmodus wird die Instruktion `IndReset` verwendet. Die logische Position der Achse kann im Zusammenhang mit dieser Instruktion geändert werden. Beispielsweise kann eine Anzahl voller Umdrehungen gelöscht werden, um ein Zurückdrehen für die nächste Bewegung zu vermeiden.

Durch ein erneutes Ausführen der Instruktion `IndAMove` (oder einer anderen `IndXMove`-Instruktion) kann die Geschwindigkeit geändert werden. Wenn eine Drehzahl in die entgegengesetzte Richtung ausgewählt wird, stoppt die Achse und beschleunigt anschließend in die neue Richtung bis auf die gewünschte Drehzahl.

Für eine schrittweise Ausführung der Instruktion befindet sich die Achse ausschließlich im unabhängigen Modus. Die Achse beginnt ihre Bewegung, wenn die nächste Instruktion ausgeführt wird, und setzt diese Bewegung für die komplette Dauer der Programmabarbeitung fort. Weitere Informationen finden Sie im *Technischen Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.99 IndAMove - Unabhängige Bewegung in absolute Position

Independent Axis

Fortsetzung

Wenn der Programmzeiger zum Programmanfang oder zu einer neuen Routine bewegt wird, werden alle Achsen automatisch in den Normalmodus versetzt, ohne dass das Messsystem geändert wird (wie bei der Instruktion `IndReset\Old`).

> **Hinweis**
>
> Eine `IndAMove`-Instruktion nach einer `IndCMove`-Operation kann dazu führen, dass die durch die Instruktion `IndCMove` ausgeführte Bewegung rückgängig gemacht wird. Um dies zu vermeiden, verwenden Sie die `IndReset`-Instruktion vor der `IndAMove`-Instruktion, oder verwenden Sie eine `IndRMove`-Instruktion.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IndAMove` sind unten aufgeführt.

Beispiel 1:

```
ActUnit Station_A;
weld_stationA;
IndAMove Station_A,1\ToAbsNum:=90,20\Ramp:=50;
ActUnit Station_B;
weld_stationB_1;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
DeactUnit Station_A;
weld_stationB_2;
```

`Station_A` wird aktiviert und der Schweißvorgang beginnt in Station A.

`Station_A` (Achse 1) wird dann in die 90-Grad-Position bewegt, während der Roboter an Station B schweißt. Die Geschwindigkeit der Achse beträgt 20 Grad/s. Die Geschwindigkeit wird mit einer auf 50 % der max. Leistung verringerten Beschleunigung/Verzögerung geändert.

Wenn Station A diese Position erreicht, wird sie deaktiviert, und es kann ein neuer Ladevorgang stattfinden, während der Roboter den Schweißvorgang zeitgleich in Station B fortsetzt.

Einschränkungen

Achsen im unabhängigen Modus können nicht manuell bewegt werden. Bei einem Versuch, die Achse manuell zu bewegen, wird die Achse nicht bewegt und es wird ein Fehler gemeldet. Mit der Instruktion `IndReset` oder durch Setzen des Programmzeigers auf „main“ können Sie den unabhängigen Modus verlassen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn sich eine Achse bei einem Stromausfall im unabhängigen Modus befindet, kann das Programm nicht neu gestartet werden. Eine Fehlermeldung wird angezeigt und das Programm muss von vorne begonnen werden.

Es ist nicht ratsam, diese Instruktion für gekoppelte Roboterhandgelenkachsen zu verwenden (siehe *Technisches Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*).

Syntax

```
IndAMove
  [MecUnit ':='] <variable (VAR) of mecunit>','
  [Axis ':='] <expression (IN) of num>
  ['\' ToAbsPos ':=' <expression (IN) of robtarget>]
  |['\' ToAbsNum ':=' <expression (IN) of num>]','
  [Speed ':='] <expression (IN) of num>
  ['\' Ramp ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Independent Axis	*Anwendungshandbuch - Steuerungssoftware IRC5*
Zurückkehren in den Normalbetrieb	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Zurücksetzen des Messsystems	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Weitere unabhängige Achsenbewegung	*IndRMove - Unabhängige Bewegung an relative Position auf Seite 279* *IndDMove - Unabhängige Delta-Position-Bewegung auf Seite 270* *IndCMove - Unabhängige kontinuierliche Bewegung auf Seite 266*
Prüfen des Geschwindigkeitsstatus für unabhängige Achsen	*IndSpeed - Unabhängiger Geschwindigkeitsstatus auf Seite 1361*
Prüfen des Positionsstatus für unabhängige Achsen	*IndInpos - Unabhängige Achse in Positionsstatus auf Seite 1359*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.100 IndCMove - Unabhängige kontinuierliche Bewegung

Verwendung

`IndCMove` (*Independent Continuous Movement*) wird verwendet, um den Modus einer Achse in "unabhängig" zu ändern und mit der kontinuierlichen Bewegung der Achse mit einer bestimmten Geschwindigkeit zu beginnen.

Eine unabhängige Achse bewegt sich unabhängig von anderen Achsen im Robotersystem. Da die Programmabarbeitung sofort fortgesetzt wird, können andere Instruktionen (einschließlich Positionierungsinstruktionen) ausgeführt werden, während sich die unabhängige Achse bewegt.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IndCMove`:

Siehe auch *Weitere Beispiele auf Seite 267*.

Beispiel 1:

```
IndCMove Station_A,2,-30.5;
```

Achse `2` von `Station_A` beginnt ihre Bewegung in negativer Richtung mit einer Geschwindigkeit von 30,5 Grad/s.

Argumente

```
IndCMove MecUnit Axis Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: num

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

`Speed`

Datentyp: `num`

Die Achsendrehzahl wird in Grad/s angegeben (in mm/s für lineare Achsen).

Die Richtung der Bewegung wird durch das Vorzeichen des Arguments für die Geschwindigkeit vorgegeben.

`[\Ramp]`

Datentyp: `num`

Verringern der Beschleunigung und der Verzögerung von maximaler Leistung (1 - 100 %, 100 % = maximale Leistung).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Wenn `IndCMove` ausgeführt wird, beginnt die angegebene Achse, sich mit der programmierten Geschwindigkeit zu bewegen. Die Richtung der Bewegung wird durch das Vorzeichen des Arguments für die Geschwindigkeit vorgegeben. Falls `\Ramp` programmiert wurde, wird auch die Beschleunigung/Verzögerung verringert.

Für die Rückkehr der Achse in den Normalmodus wird die Instruktion `IndReset` verwendet. Die logische Position der Achse kann im Zusammenhang mit dieser Instruktion geändert werden. Beispielsweise kann eine Anzahl voller Umdrehungen gelöscht werden, um ein Zurückdrehen für die nächste Bewegung zu vermeiden.

Die Geschwindigkeit kann geändert werden, indem die Instruktion `IndCMove` ein weiteres Mal ausgeführt wird. Wenn eine Geschwindigkeit in die entgegengesetzte Richtung angefordert wird, stoppt die Achse und beschleunigt anschließend in die neue Richtung bis auf die gewünschte Geschwindigkeit. Um die Achse anzuhalten, kann für das Geschwindigkeitsargument 0 verwendet werden. So bleibt die Achse auch weiterhin im unabhängigen Modus.

Bei der schrittweisen Ausführung der Instruktion befindet sich die Achse ausschließlich im unabhängigen Modus. Die Achse beginnt ihre Bewegung, wenn die nächste Instruktion ausgeführt wird, und setzt diese Bewegung für die komplette Dauer der Programmabarbeitung fort. Weitere Informationen finden Sie im *Technischen Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*

Wenn der Programmzeiger zum Programmanfang oder zu einer neuen Routine bewegt wird, werden alle Achsen automatisch in den Normalmodus versetzt, ohne dass das Messsystem geändert wird (wie bei der Instruktion `IndReset\Old`).

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IndCMove` sind unten aufgeführt.

```
IndCMove Station_A,2,20;
WaitUntil IndSpeed(Station_A,2 \InSpeed) = TRUE;
WaitTime 0.2;
MoveL p10, v1000, fine, tool1;
IndCMove Station_A,2,-10\Ramp:=50;
MoveL p20, v1000, z50, tool1;
IndRMove Station_A,2 \ToRelPos:=p1 \Short,10;
MoveL p30, v1000, fine, tool1;
WaitUntil IndInpos(Station_A,2 ) = TRUE;
WaitTime 0.2;
IndReset Station_A,2 \RefPos:=p40\Short;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.100 IndCMove - Unabhängige kontinuierliche Bewegung

Independent Axis

Fortsetzung

```
MoveL p40, v1000, fine, tool1;
```

Achse 2 von Station_A beginnt ihre Bewegung in positiver Richtung mit einer Geschwindigkeit von 20 Grad/s. Wenn diese Achse die gewünschte Geschwindigkeit erreicht hat, beginnen sich die Roboterachsen zu bewegen.

Wenn der Roboter die Position p10 erreicht, ändert die externe Achse ihre Richtung und dreht sich mit einer Geschwindigkeit von 10 Grad/s. Die Geschwindigkeitsänderung erfolgt mit einer auf 50 % der maximalen Leistung verringerten Beschleunigung/Verzögerung. Gleichzeitig bewegt sich der Roboter in Richtung p20.

Achse 2 von Station_A wird dann so schnell wie möglich in der aktuellen Umdrehung an Position p1 angehalten.

Wenn Achse 2 diese Position erreicht hat und der Roboter an Position p30 angehalten wurde, wechselt Achse 2 wieder zurück in den Normalmodus. Der Offset des Messsystems für die Achse wird um eine ganzzahlige Anzahl an Achsumdrehungen geändert, damit die tatsächliche Position möglichst nahe an Position p40 liegt.

Wenn der Roboter anschließend in die Position p40 gebracht wird, wird Achse 2 von Station_A mithilfe der Instruktion MoveL p40 auf kürzestem Weg in die Position p40 gebracht (max. ±180 Grad).

Einschränkungen

Die Auflösung der Achsenposition verschlechtert sich, je weiter sie sich von ihrer logischen Nullstellung entfernt (normalerweise die Mitte des Arbeitsbereichs). Um erneut eine hohe Auflösung zu erhalten, kann der logische Arbeitsbereich mithilfe der Instruktion IndReset auf null gesetzt werden. Weitere Informationen finden Sie im *Technischen Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*

Achsen im unabhängigen Modus können nicht manuell bewegt werden. Bei einem Versuch, die Achse manuell zu bewegen, wird die Achse nicht bewegt und es wird ein Fehler gemeldet. Mit der Instruktion IndReset oder durch Setzen des Programmzeigers auf „main“ können Sie den unabhängigen Modus verlassen.

Wenn sich die Achse bei einem Stromausfall im unabhängigen Modus befindet, kann das Programm nicht neu gestartet werden. Eine Fehlermeldung wird angezeigt und das Programm muss von vorne begonnen werden.

Es ist nicht ratsam, diese Instruktion für gekoppelte Roboterhandgelenkachsen zu verwenden (siehe *RAPID Referenzhandbuch - RAPID-Übersicht*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*).

Syntax

```
IndCMove
  [MecUnit ':='] <variable (VAR) of mecunit>','
  [Axis ':='] <expression (IN) of num>','
  [Speed ':='] <expression (IN) of num>
  ['\' Ramp ':=' <expression (IN) of num>]';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Independent Axis	*Anwendungshandbuch - Steuerungssoftware IRC5*
Zurückkehren in den Normalbetrieb	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Zurücksetzen des Messsystems	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Weitere unabhängige Achsenbewegung	*IndAMove - Unabhängige Bewegung in absolute Position auf Seite 262* *IndRMove - Unabhängige Bewegung an relative Position auf Seite 279* *IndDMove - Unabhängige Delta-Position-Bewegung auf Seite 270*
Prüfen des Geschwindigkeitsstatus für unabhängige Achsen	*IndSpeed - Unabhängiger Geschwindigkeitsstatus auf Seite 1361*
Prüfen des Positionsstatus für unabhängige Achsen	*IndInpos - Unabhängige Achse in Positionsstatus auf Seite 1359*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.101 IndDMove - Unabhängige Delta-Position-Bewegung

Verwendung

`IndDMove`(*Independent Delta Movement*) wird verwendet, um den Modus einer Achse in „unabhängig“ zu ändern und die Achse in eine bestimmte Entfernung zu bringen.

Eine unabhängige Achse bewegt sich unabhängig von anderen Achsen im Robotersystem. Da die Programmabarbeitung sofort fortgesetzt wird, können andere Instruktionen (einschließlich Positionierungsinstruktionen) ausgeführt werden, während sich die unabhängige Achse bewegt.

Wenn die Achse an eine bestimmte Position bewegt werden soll, muss stattdessen die Instruktion `IndAMove` oder `IndRMove` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IndDMove`:

Siehe auch *Weitere Beispiele auf Seite 271*.

Beispiel 1:

```
IndDMove Station_A,2,-30,20;
```

Achse `2` von `Station_A` wird mit einer Geschwindigkeit von `20` Grad/s um `30` Grad in negative Richtung bewegt.

Argumente

```
IndDMove MecUnit Axis Delta Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

`Delta`

Datentyp: `num`

Die Distanz in Grad (mm für lineare Achsen), über die die aktuelle Achse bewegt wird. Das Vorzeichen gibt die Richtung der Bewegung an.

`Speed`

Datentyp: `num`

Die Achsendrehzahl wird in Grad/s angegeben (in mm/s für lineare Achsen).

`[ \Ramp ]`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Verringern der Beschleunigung und der Verzögerung von maximaler Leistung (1 - 100 %, 100 % = maximale Leistung).

Programmabarbeitung

Wenn `IndDMove` ausgeführt wird, bewegt sich die angegebene Achse mit der programmierten Geschwindigkeit zu dem angegebenen Abstand. Die Richtung der Bewegung wird durch das Vorzeichen des Arguments `Delta` vorgegeben. Falls `\Ramp` programmiert wurde, wird auch die Beschleunigung/Verzögerung verringert.

Wenn sich die Achse bewegt, wird die neue Position aus der aktuellen Position der Achse berechnet, wenn die Instruktion `IndDMove` ausgeführt wird. Wenn die Instruktion `IndDMove` mit der Distanz 0 ausgeführt wird und sich die Achse bereits bewegt, hält die Achse an und bewegt sich zurück zu der Position, an der sie sich zum Zeitpunkt der Ausführung der Anweisung befunden hat.

Für die Rückkehr der Achse in den Normalmodus wird die Instruktion `IndReset` verwendet. Die logische Position der Achse kann im Zusammenhang mit dieser Instruktion geändert werden. Beispielsweise kann eine Anzahl voller Umdrehungen für die Position gelöscht werden, um ein Zurückdrehen für die nächste Bewegung zu vermeiden.

Durch ein erneutes Ausführen der Instruktion `IndDMove` (oder einer anderen `IndXMove`-Instruktion) kann die Geschwindigkeit geändert werden. Wenn eine Drehzahl in die entgegengesetzte Richtung ausgewählt wird, stoppt die Achse und beschleunigt anschließend in die neue Richtung bis auf die gewünschte Drehzahl.

Bei der schrittweisen Ausführung der Instruktion befindet sich die Achse ausschließlich im unabhängigen Modus. Die Achse beginnt ihre Bewegung, wenn die nächste Instruktion ausgeführt wird, und setzt diese Bewegung für die komplette Dauer der Programmabarbeitung fort. Weitere Informationen finden Sie im *Technischen Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*

Wenn der Programmzeiger zum Programmanfang oder zu einer neuen Routine bewegt wird, werden alle Achsen automatisch in den Normalmodus versetzt, ohne dass das Messsystem geändert wird (wie bei der Instruktion `IndReset \Old`).

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IndDMove` sind unten aufgeführt.

Beispiel 1:

```
IndAMove ROB_1,6\ToAbsNum:=90,20;
WaitUntil IndInpos(ROB_1,6) = TRUE;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.101 IndDMove - Unabhängige Delta-Position-Bewegung

Independent Axis

Fortsetzung

```
WaitTime 0.2;
IndDMove Station_A,2,-30,20;
WaitUntil IndInpos(ROB_1,6) = TRUE;
WaitTime 0.2;
IndDMove ROB_1,6,400,20;
```

Achse 6 des Roboters wird an folgende Positionen bewegt:

- 90 Grad
- 60 Grad
- 460 Grad (1 Umdrehung + 100 Grad)

Einschränkungen

Achsen im unabhängigen Modus können nicht manuell bewegt werden. Bei einem Versuch, die Achse manuell zu bewegen, wird die Achse nicht bewegt und es wird ein Fehler gemeldet. Mit der Instruktion `IndReset` oder durch Setzen des Programmzeigers auf „main“ können Sie den unabhängigen Modus verlassen.

Wenn sich die Achse bei einem Stromausfall im unabhängigen Modus befindet, kann das Programm nicht neu gestartet werden. Eine Fehlermeldung wird angezeigt und das Programm muss von vorne begonnen werden.

Es ist nicht ratsam, diese Instruktion für gekoppelte Roboterhandgelenkachsen zu verwenden (siehe *Technisches Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*).

Syntax

```
IndDMove
  [MecUnit ':='] <variable (VAR) of mecunit>','
  [Axis ':='] <expression (IN) of num>','
  [Delta ':='] <expression (IN) of num>','
  [Speed '='] <expression (IN) of num>
  ['\' Ramp ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Independent Axis	*Anwendungshandbuch - Steuerungssoftware IRC5*
Zurückkehren in den Normalbetrieb	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Zurücksetzen des Messsystems	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Weitere unabhängige Achsenbewegung	*IndAMove - Unabhängige Bewegung in absolute Position auf Seite 262* *IndRMove - Unabhängige Bewegung an relative Position auf Seite 279* *IndCMove - Unabhängige kontinuierliche Bewegung auf Seite 266*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Prüfen des Geschwindigkeitsstatus für unabhängige Achsen	*IndSpeed - Unabhängiger Geschwindigkeitsstatus auf Seite 1361*
Prüfen des Positionsstatus für unabhängige Achsen	*IndInpos - Unabhängige Achse in Positionsstatus auf Seite 1359*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.102 IndReset - Unabhängiges Zurücksetzen

Verwendung

`IndReset` (*Independent Reset*) wird verwendet, um eine unabhängige Achse in den Normalmodus zurückzusetzen. Gleichzeitig kann das Messsystem für Rotationsachsen um eine Anzahl an Achsenumdrehungen bewegt werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IndReset`:

Siehe auch *Weitere Beispiele auf Seite 276*.

```
IndCMove Station_A,2,5;
MoveL *,v1000,fine,tool1;
IndCMove Station_A,2,0;
WaitUntil IndSpeed(Station_A,2\ZeroSpeed);
WaitTime 0.2
IndReset Station_A,2;
```

Achse `2` von `Station_A` wird zuerst im unabhängigen Modus bewegt und anschließend in den Normalmodus zurückgesetzt. Die Achse behält dabei ihre Position bei.

> **Hinweis**
>
> Beachten Sie, dass sich die aktuelle unabhängige Achse sowie die Achsen im Normalmodus nicht bewegen sollten, während die Instruktion `IndReset` ausgeführt wird. Der Grund hierfür ist, dass es sich bei der vorherigen Position um einen Stopppunkt handelt und die Instruktion `IndCMove` mit der Geschwindigkeit null ausgeführt wird. Darüber hinaus wird eine Pause von 0,2 Sekunden verwendet, um sicherzustellen, dass der korrekte Status erreicht wurde.

Argumente

```
IndReset MecUnit Axis [\RefPos] | [\RefNum] [\Short] | [\Fwd]
     |[\Bwd] | \Old]
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

`[ \RefPos ]`

Reference Position

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `robtarget`

Die Referenzposition der Achse wird unter `robtarget` angegeben. Es wird nur die Komponente für diese bestimmte `Axis` verwendet. Die Position muss innerhalb des normalen Arbeitsbereichs liegen.

Für Roboterachsen muss stattdessen das Argument `\RefNum` verwendet werden.

Das Argument kann nur zusammen mit dem Argument `\Short`, `\Fwd` oder `\Bwd` definiert werden. Zusammen mit dem Argument `\Old` ist es nicht zulässig.

`[ \RefNum ]`

Reference Numeric value

Datentyp: `num`

Die Referenzposition der Achse ist in Grad definiert (in mm für lineare Achsen). Die Position muss innerhalb des normalen Arbeitsbereichs liegen.

Das Argument kann nur zusammen mit dem Argument `\Short`, `\Fwd` oder `\Bwd` definiert werden. Zusammen mit dem Argument `\Old` ist es nicht zulässig.

Diese Funktion entspricht derjenigen von `\RefPos`, mit dem Unterschied, dass die Position als numerischer Wert definiert ist, wodurch eine manuelle Änderung der Position problemlos möglich ist.

`[ \Short ]`

Datentyp: `switch`

Das Messsystem wird achsseitig um eine ganzzahlige Anzahl an Umdrehungen geändert, damit sich die Achse so nahe wie möglich an der unter `\RefPos` oder `\RefNum` angegebenen Position befindet. Wenn nach der Instruktion `IndReset` eine Positionierungsinstruktion mit derselben Position ausgeführt wird, bewegt sich die Achse auf dem kürzesten Weg (weniger als ±180 Grad), um diese Position zu erreichen.

`[ \Fwd ]`

Forward

Datentyp: `switch`

Das Messsystem wird achsseitig um eine ganzzahlige Anzahl an Umdrehungen geändert, damit sich die Referenzposition auf der positiven Seite der angegebenen `\RefPos`- oder `\RefNum`-Position befindet. Wenn nach der Instruktion `IndReset` eine Positionierungsinstruktion mit derselben Position ausgeführt wird, bewegt sich die Achse in eine positive Richtung von weniger als 360 Grad, um diese Position zu erreichen.

`[ \Bwd ]`

Backward

Datentyp: `switch`

Das Messsystem wird achsseitig um eine ganzzahlige Anzahl an Umdrehungen geändert, damit sich die Referenzposition auf der negativen Seite der angegebenen `\RefPos`- oder `\RefNum`-Position befindet. Wenn nach der Instruktion `IndReset` eine Positionierungsinstruktion mit derselben Position ausgeführt wird, bewegt

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.102 IndReset - Unabhängiges Zurücksetzen
Independent Axis
Fortsetzung

sich die Achse in eine negative Richtung von weniger als 360 Grad, um diese Position zu erreichen.

`[ \Old ]`

Datentyp: `switch`

Behält die alte Position bei.

> **Hinweis**
>
> Beachten Sie, dass sich die Auflösung an weit vom Nullpunkt entfernten Positionen verschlechtert.
>
> Wenn die Argumente `\Short`, `\Fwd`, `\Bwd` oder `\Old` nicht angegeben werden, wird `\Old` als Standardwert verwendet.

Programmabarbeitung

Wenn die Instruktion `IndReset` ausgeführt wird, wird eine unabhängige Achse in den Normalmodus zurückgesetzt. Gleichzeitig kann das Messsystem für die Achse um eine ganzzahlige Anzahl an Achsenumdrehungen bewegt werden.

Die Instruktion kann auch im Normalmodus verwendet werden, um das Messsystem zu ändern.

> **Hinweis**
>
> Beachten Sie, dass die Position nur für das Einstellen des Messsystems verwendet wird. Die Achse wird sich nicht an diese Position bewegen.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.
`ERR_AXIS_MOVING`	Die Achse bewegt sich.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IndReset` sind unten aufgeführt.

Beispiel 1:

```
IndAMove Station_A,1\ToAbsNum:=750,50;
WaitUntil IndInpos(Station_A,1);
WaitTime 0.2;
IndReset Station_A,1 \RefNum:=0 \Short;
IndAMove Station_A,1\ToAbsNum:=750,50;
WaitUntil IndInpos(Station_A,1);
WaitTime 0.2;
IndReset Station_A,1 \RefNum:=300 \Short;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Achse `1` in `Station_A` wird zunächst unabhängig in die `750`-Grad-Position (2 Umdrehungen plus 30 Grad) gebracht. Wenn die Achse in den Normalmodus versetzt wird, wird gleichzeitig die logische Position auf 30 Grad eingestellt.

Achse `1` in `Station_A` wird anschließend in die `750`-Grad-Position (2 Umdrehungen plus 30 Grad) gebracht. Wenn die Achse in den Normalmodus versetzt wird, wird gleichzeitig die logische Position auf 390 Grad (1 Umdrehung plus 30 Grad) eingestellt.

Einschränkungen

Die Instruktion darf nur ausgeführt werden, wenn alle aktiven Achsen im Normalmodus stillstehen. Alle aktiven Achsen in jeder mechanischen Einheit, die mit derselben Bewegungsplanung verbunden ist, müssen still stehen. Auch die Achse im unabhängigen Modus, die in den Normalmodus zurückgesetzt werden soll, darf sich nicht bewegen. Bei Achsen im Normalmodus lässt sich dies erreichen, indem eine Bewegungsinstruktion mit dem Argument `fine` ausgeführt wird. Die unabhängige Achse kann mithilfe der Instruktion `IndCMove` mit `Speed:=0` (gefolgt von einer Wartezeit von 0,2 Sekunden), `IndRMove`, `IndAMove` oder `IndDMove` angehalten werden.

Die Auflösung der Positionen wird verringert, wenn die Entfernung von der logischen Position 0 zunimmt. Eine Achse, die stufenweise in immer größerer Entfernung von Position A rotiert, sollte daher mit der Instruktion `IndReset` und einem anderen Argument als `\Old` auf null gesetzt werden.

Das Messsystem kann für lineare Achsen nicht geändert werden.

Um nach Ausführung der Instruktion `IndReset` den einwandfreien Start einer Achse mit einem relativen Messsystem (Synchronisierungsschalter) sicherzustellen, muss nach der Ausführung der Instruktion `IndReset` eine zeitliche Verzögerung von 0,12 Sekunden hinzugefügt werden.

Nur Roboterachse 6 kann als unabhängige Achse verwendet werden. Die Instruktion `IndReset` kann bei den Modellen IRB 1600, 2600 und 4600 (nicht bei der ID-Version) auch für Achse 4 verwendet werden. Wenn `IndReset` für Roboterachse 4 verwendet wird, darf sich Achse 6 nicht im unabhängigen Modus befinden.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (`zonedata fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`IndReset` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

`IndReset` wechselt nur in den Status "Unabhängig" für eine Achse. Es kann nicht zum Anhalten von "Independent movement" (unabhängige Bewegung) verwendet werden. Um eine unabhängige Bewegung anzuhalten, muss eine Stopp-Bedingung eintreten oder der Benutzer muss z. B. den PZ auf Main verschieben.

Syntax

```
IndReset
  [MecUnit ':='] <variable (VAR) of mecunit>','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.102 IndReset - Unabhängiges Zurücksetzen

Independent Axis

Fortsetzung

```
[Axis ':='] <expression (IN) of num>
['\' RefPos ':=' <expression (IN) of robtarget>]|
['\' RefNum ':=' <expression (IN) of num>]
['\' Short] | ['\' Fwd] | ['\' Bwd] | ['\' Old]';'
```

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Independent Axis	*Anwendungshandbuch - Steuerungssoftware IRC5*
Ändern des Modus einer Achse in "unabhängig"	*IndAMove - Unabhängige Bewegung in absolute Position auf Seite 262* *IndCMove - Unabhängige kontinuierliche Bewegung auf Seite 266* *IndDMove - Unabhängige Delta-Position-Bewegung auf Seite 270* *IndRMove - Unabhängige Bewegung an relative Position auf Seite 279*
Prüfen des Geschwindigkeitsstatus für unabhängige Achsen	*IndSpeed - Unabhängiger Geschwindigkeitsstatus auf Seite 1361*
Prüfen des Positionsstatus für unabhängige Achsen	*IndInpos - Unabhängige Achse in Positionsstatus auf Seite 1359*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.103 IndRMove - Unabhängige Bewegung an relative Position

Verwendung

`IndRMove` (*Independent Relative Movement*) wird verwendet, um den Modus einer Rotationsachse in "unabhängig" zu ändern und die Achse mit einer Umdrehung in eine bestimmte Position zu bringen.

Eine unabhängige Achse bewegt sich unabhängig von anderen Achsen im Robotersystem. Da die Programmabarbeitung sofort fortgesetzt wird, können andere Instruktionen (einschließlich Positionierungsinstruktionen) ausgeführt werden, während sich die unabhängige Achse bewegt.

Wenn die Achse in eine absolute Position (mehrere Umdrehungen) gebracht werden soll oder es sich um eine lineare Achse handelt, wird stattdessen die Instruktion `IndAMove` verwendet. Wenn die Bewegung in einem bestimmten Abstand von der aktuellen Position entfernt erfolgen soll, muss die Instruktion `IndDMove` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IndRMove`:

Siehe auch *Weitere Beispiele auf Seite 281*.

Beispiel 1:

```
IndRMove Station_A,2\ToRelPos:=p5 \Short,20;
```

Achse `2` von `Station_A` wird auf dem kürzesten Weg in einer Umdrehung an Position `p5` bewegt (max. Rotation ± 180 Grad), und zwar mit einer Geschwindigkeit von `20` Grad/s.

Argumente

```
IndRMove MecUnit Axis [\ToRelPos] | [\ToRelNum] [\Short] | [\Fwd]
     | [\Bwd] Speed [\Ramp]
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

`[ \ToRelPos ]`

To Relative Position

Datentyp: `robtarget`

Die Achsenposition wird unter `robtarget` angegeben. Es wird nur die Komponente für diese bestimmte `Axis` verwendet. Der Wert wird als Positionswert in Grad

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.103 IndRMove - Unabhängige Bewegung an relative Position
Independent Axis
Fortsetzung

innerhalb einer Achsenumdrehung verwendet. Das bedeutet, dass sich die Achse um weniger als eine Umdrehung dreht.

Es hat Auswirkungen auf die Achsenposition, wenn die Position der Achse mithilfe der Instruktion `EOffsSet` oder `EOffsOn` geändert wird.

Für Roboterachsen muss stattdessen das Argument `\ToRelNum` verwendet werden.

`[ \ToRelNum ]`

To Relative Numeric value

Datentyp: `num`

Die Achsenposition in Grad.

Bei Verwendung dieses Arguments haben Positionsänderungen mithilfe von `EOffsSet` oder `PDispOn` KEINE Auswirkungen auf die Position.

Diese Funktion entspricht derjenigen von `\ToRelPos`, mit dem Unterschied, dass die Position als numerischer Wert definiert ist, wodurch eine manuelle Änderung der Position problemlos möglich ist.

`[ \Short ]`

Datentyp: `switch`

Die Achse wird auf dem kürzesten Weg an die neue Position bewegt. Das bedeutet, dass die maximale Drehung in eine beliebige Richtung 180 Grad beträgt. Die Bewegungsrichtung hängt daher von der aktuellen Position der Achse ab.

`[ \Fwd ]`

Forward

Datentyp: `switch`

Die Achse wird in positiver Richtung an die neue Position bewegt. Das bedeutet, dass die maximale Drehung 360 Grad beträgt und immer in positiver Richtung erfolgt (erhöhter Positionswert).

`[ \Bwd ]`

Backward

Datentyp: `switch`

Die Achse wird in negativer Richtung an die neue Position bewegt. Das bedeutet, dass die maximale Drehung 360 Grad beträgt und immer in negativer Richtung erfolgt (verringerter Positionswert).

Wenn das Argument `\Short`, `\Fwd` oder `\Bwd` weggelassen wird, wird `\Short` als Standardwert verwendet.

`Speed`

Datentyp: `num`

Die Achsengeschwindigkeit in Grad/s.

`[ \Ramp ]`

Datentyp: `num`

Verringern der Beschleunigung und der Verzögerung von maximaler Leistung (1 - 100 %, 100 % = maximale Leistung).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Wenn `IndRMove` ausgeführt wird, bewegt sich die angegebene Achse mit der programmierten Geschwindigkeit in die angegebene Achsenposition, jedoch nur um maximal eine Umdrehung. Falls `\Ramp` programmiert wurde, wird auch die Beschleunigung/Verzögerung verringert.

Für die Rückkehr der Achse in den Normalmodus wird die Instruktion `IndReset` verwendet. Die logische Position der Achse kann im Zusammenhang mit dieser Instruktion geändert werden. Beispielsweise kann eine Anzahl voller Umdrehungen für die Position gelöscht werden, um ein Zurückdrehen für die nächste Bewegung zu vermeiden.

Durch ein erneutes Ausführen der Instruktion `IndRMove` (oder einer anderen `IndXMove`-Instruktion) kann die Geschwindigkeit geändert werden. Wenn eine Drehzahl in die entgegengesetzte Richtung ausgewählt wird, stoppt die Achse und beschleunigt anschließend in die neue Richtung bis auf die gewünschte Drehzahl.

Bei der schrittweisen Ausführung der Instruktion befindet sich die Achse ausschließlich im unabhängigen Modus. Die Achse beginnt ihre Bewegung, wenn die nächste Instruktion ausgeführt wird, und setzt diese Bewegung für die komplette Dauer der Programmabarbeitung fort. Weitere Informationen finden Sie unter *RAPID Referenzhandbuch - RAPID-Übersicht*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*

Wenn der Programmzeiger zum Programmanfang oder zu einer neuen Routine bewegt wird, werden alle Achsen automatisch in den Normalmodus versetzt, ohne dass das Messsystem geändert wird (wie bei der Instruktion `IndReset \Old`).

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IndRMove` sind unten aufgeführt.

Beispiel 1:

```
IndRMove Station_A,1\ToRelPos:=p5 \Fwd,20\Ramp:=50;
```

Achse `1` von `Station_A` beginnt ihre Bewegung in positiver Richtung zu Position `p5` in einer Umdrehung (max. Rotation 360 Grad), und zwar mit einer Geschwindigkeit von `20` Grad/s. Die Drehzahländerung wird mit einer auf `50` % der maximalen Leistung reduzierten Beschleunigung/Verzögerung durchgeführt.

```
IndAMove Station_A,1\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
IndRMove Station_A,1\ToRelNum:=80 \Fwd,20;
WaitTime 0.2;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.103 IndRMove - Unabhängige Bewegung an relative Position

Independent Axis

Fortsetzung

```
WaitTime 0.2;
IndRMove Station_A,1\ToRelNum:=50 \Bwd,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndRMove Station_A,1\ToRelNum:=150 \Short,20;
WaitUntil IndInpos(Station_A,1 ) = TRUE;
WaitTime 0.2;
IndAMove Station_A,1\ToAbsNum:=10,20;
```

Achse `1` von `Station_A` wird an folgende Positionen bewegt:

- 90 Grad
- 440 Grad (1 Umdrehung +80 Grad)
- 410 Grad (1 Umdrehung 50 Grad)
- 510 Grad (1 Umdrehung 150 Grad)
- 10 Grad

Einschränkungen

Achsen im unabhängigen Modus können nicht manuell bewegt werden. Bei einem Versuch, die Achse manuell zu bewegen, wird die Achse nicht bewegt und es wird ein Fehler gemeldet. Mit der Instruktion `IndReset` oder durch Setzen des Programmzeigers auf „main“ können Sie den unabhängigen Modus verlassen.

Wenn sich die Achse bei einem Stromausfall im unabhängigen Modus befindet, kann das Programm nicht neu gestartet werden. Eine Fehlermeldung wird angezeigt und das Programm muss von vorne begonnen werden.

Es ist nicht ratsam, diese Instruktion für gekoppelte Roboterhandgelenkachsen zu verwenden (siehe *RAPID Referenzhandbuch - RAPID-Übersicht*, Abschnitt *Bewegung und E/A-Prinzipien - Positionierung während der Abarbeitung von Programmen - Unabhängige Achsen*).

Syntax

```
IndRMove
  [MecUnit ':='] <variable (VAR) of mecunit>','
  [Axis ':='] <expression (IN) of num>
  ['\' ToRelPos ':=' <expression (IN) of robtargets>]
  |['\' ToRelNum ':=' <expression (IN) of num>]
  ['\' Short] | ['\' Fwd] | ['\' Bwd]','
  [Speed ':='] <expression (IN) of num>
  ['\' Ramp ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Independent Axis	*Anwendungshandbuch - Steuerungssoftware IRC5*
Zurückkehren in den Normalbetrieb	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Zurücksetzen des Messsystems	*IndReset - Unabhängiges Zurücksetzen auf Seite 274*
Weitere unabhängige Achsenbewegung	*IndAMove - Unabhängige Bewegung in absolute Position auf Seite 262* *IndDMove - Unabhängige Delta-Position-Bewegung auf Seite 270* *IndCMove - Unabhängige kontinuierliche Bewegung auf Seite 266*
Prüfen des Geschwindigkeitsstatus für unabhängige Achsen	*IndSpeed - Unabhängiger Geschwindigkeitsstatus auf Seite 1361*
Prüfen des Positionsstatus für unabhängige Achsen	*IndInpos - Unabhängige Achse in Positionsstatus auf Seite 1359*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.104 InitSuperv - Zurücksetzen der gesamten CAP-Überwachung

Verwendung

`InitSuperv` wird verwendet, um die CAP-Überwachung zu initiieren. Das bedeutet, dass alle Überwachungslisten gelöscht und alle E/A-Abonnements entfernt werden.

Beispiel

```
PROC main()
  InitSuperv;
  SetupSuperv diWR_EST, ACT,SUPERV_MAIN;
  SetupSuperv diGA_EST, ACT,SUPERV_MAIN;
  CapL p2, v100, cdata1, weavestart, weave,fine, tWeldGun;
ENDPROC
```

`InitSuperv` wird verwendet, um alle Überwachungslisten vor der Einrichtung einer neuen Überwachung zu löschen.

Einschränkungen

Die `InitSuperv`-Instruktion darf nur einmal ausgeführt werden z.B. vor dem Anlaufen.

Syntax

```
InitSuperv ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instruktion `SetupSuperv`	*SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP auf Seite 725*
Instruktion `RemoveSuperv`	*RemoveSuperv Bedingung für ein Signal entfernen auf Seite 607*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.105 InvertDO - Invertieren des Werts eines digitalen Ausgangssignals

Verwendung

`InvertDO` (*Invert Digital Output*) invertiert den Wert eines digitalen Ausgangssignals (0 -> 1 und 1 -> 0).

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `InvertDO`:

Beispiel 1:

```
InvertDO do15;
```

Der aktuelle Wert des Signals `do15` wird invertiert .

Argumente

```
InvertDO Signal
```

`Signal`

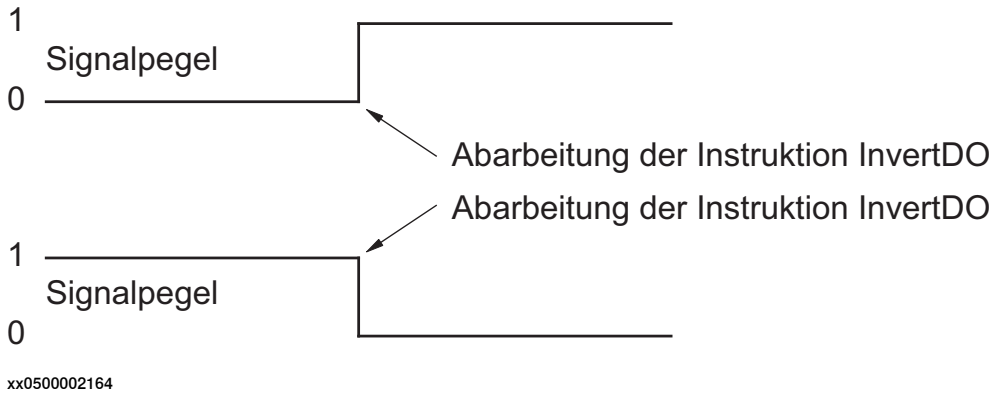
Datentyp: `signaldo`

Der Name des zu invertierenden Signals.

Programmabarbeitung

Der aktuelle Wert des Signals wird invertiert (siehe nachfolgende Abbildung).

Die Abbildung unten zeigt die Invertierung eines digitalen Ausgangssignals.



xx0500002164

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden (gilt nur für ICI-Feldbus).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.105 InvertDO - Invertieren des Werts eines digitalen Ausgangssignals

RobotWare - OS

Fortsetzung

Syntax

```
InvertDO
  [Signal ':='] <variable (VAR) of signaldo>';'
```

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.106 IOBusStart - Start of I/O network

Verwendung

`IOBusStart` wird zum Starten eines bestimmten E/A-Netzwerks verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IOBusStart`:

Beispiel 1:

```
IOBusStart "IBS";
```

Die Instruktion startet das E/A-Netzwerk mit dem Namen `IBS`.

Argumente

```
IOBusStart BusName
```

`BusName`

Datentyp: `string`

Der Name des zu startenden E/A-Netzwerks.

Programmabarbeitung

Startet das E/A-Netzwerk mit dem im Parameter `BusName` angegebenen Namen.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_NAME_INVALID`	Der Name des E/A-Netzwerks existiert nicht.

Syntax

```
IOBusStart
  [ BusName ':=' ] < expression (IN) of string>';'
```

Weitere Informationen

Information über	Siehe
Abrufen des E/A-Netzwerkstatus	*IOBusState - Abrufen des aktuellen Status des E/A-Netzwerks auf Seite 288*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.107 IOBusState - Abrufen des aktuellen Status des E/A-Netzwerks

Verwendung

Mit `IOBusState` wird der Status eines bestimmten E/A-Netzwerks gelesen. Der physische und logische Status definieren den Status eines E/A-Netzwerks.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `IOBusState`:

Beispiel 1:

```
VAR busstate bstate;

IOBusState "IBS", bstate \Phys;
TEST bstate
CASE IOBUS_PHYS_STATE_RUNNING:
  ! Possible to access the signals on the IBS bus
DEFAULT:
  ! Actions for not up and running IBS bus
ENDTEST
```

Die Instruktion gibt den physischen Status des E/A-Netzwerks von `IBS` in der Variablen `bstate` vom Typ `busstate` zurück.

Beispiel 2:

```
VAR busstate bstate;

IOBusState "IBS", bstate \Logic;
TEST bstate
CASE IOBUS_LOG_STATE_STARTED:
  ! The IBS bus is started
DEFAULT:
  ! Actions for stopped IBS bus
ENDTEST
```

Die Instruktion gibt den logischen Status des E/A-Netzwerks von `IBS` in der Variablen `bstate` vom Typ `busstate` zurück.

Argumente

```
IOBusState BusName State [\Phys] | [\Logic]
```

`BusName`

Datentyp: `string`

Der Name des E/A-Netzwerks, dessen Status abgerufen werden soll.

`State`

Datentyp: `busstate`

Die Variable, in der der Status des E/A-Netzwerks zurückgegeben wird. Siehe weiter unten bei der Programmabarbeitung die vordefinierten Daten vom Typ `busstate`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\Phys]`

Physical

Datentyp: `switch`

Bei Verwendung dieses Parameters wird der physische Status des E/A-Netzwerks gelesen.

`[\Logic]`

Logical

Datentyp: `switch`

Bei Verwendung dieses Parameters wird der logische Status des E/A-Netzwerks gelesen.

Programmabarbeitung

Gibt im Parameter `State` den Status des im Parameter `BusName` angegebenen E/A-Netzwerks zurück.

Die logischen E/A-Netzwerkstatus beschreiben den Status, den der Benutzer für den Bus anweisen kann. Der Status des E/A-Busses bei Verwendung des optionalen Arguments `\Logic` wird in der folgenden Tabelle beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
10	IOBUS_LOG_STATE_STOPPED	Der Bus wird aufgrund eines Fehlers angehalten [2]
11	IOBUS_LOG_STATE_STARTED	Bus wird gestartet [1]

Der physische E/A-Netzwerkstatus beschreibt den Status, den der Feldbustreiber für den Bus anweisen kann. Der Status des E/A-Netzwerks bei Verwendung des optionalen Arguments `\Phys` wird in der folgenden Tabelle beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
20	IOBUS_PHYS_STATE_HALTED	Bus wird angehalten [3]
21	IOBUS_PHYS_STATE_RUNNING	Bus ist bereit und in Betrieb [1]
22	IOBUS_PHYS_STATE_ERROR	Bus funktioniert nicht [2]
23	IOBUS_PHYS_STATE_STARTUP	Bus ist im Startmodus und kommuniziert mit keinen E/A-Geräten.
24	IOBUS_PHYS_STATE_INIT	Bus wurde nur erstellt [3]

Hinweis

Der Status des E/A-Netzwerks, wenn keines der optionalen Argumente `\Phys` oder `\Logic` verwendet wird, wird in der folgenden Tabelle beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
0	BUSSTATE_HALTED	Bus wird angehalten [3]
1	BUSSTATE_RUN	Bus ist bereit und in Betrieb [1]
2	BUSSTATE_ERROR	Bus funktioniert nicht [2]
3	BUSSTATE_STARTUP	Bus ist im Startmodus und kommuniziert mit keinen E/A-Geräten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.107 IOBusState - Abrufen des aktuellen Status des E/A-Netzwerks
RobotWare - OS
Fortsetzung

Rückgabewert	Symbolische Konstante	Kommentar
4	BUSSTATE_INIT	Bus wurde nur erstellt [3]

[1] Wenn das E/A-Netzwerks in Betrieb ist, kann der im Argument `State` in der Instruktion `IOBusState` zurückgegebene Status `IOBUS_LOG_STATE_STARTED`, `IOBUS_PHYS_STATE_RUNNING` oder `BUSSTATE_RUN` lauten, je nachdem, ob in `IOBusState` optionale Parameter verwendet werden.

[2] Wenn das E/A-Netzwerk aufgrund eines Fehlers angehalten wird, kann der im Argument `State` zurückgegebene Status `IOBUS_LOG_STATE_STOPPED`, `IOBUS_PHYS_STATE_ERROR` oder `BUSSTATE_ERROR` lauten, je nachdem, ob in `IOBusState` optionale Parameter verwendet werden.

[3] Mit der aktuellen Version von Robotware - OS kann dieser Status nicht im RAPID-Programm abgerufen werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_NAME_INVALID`	Der Name des E/A-Netzwerks existiert nicht.

Syntax

```
IOBusState
  [ BusName ':=' ] < expression (IN) of string> ','
  [ State ':=' ] < variable (VAR) of busstate>
  [ '\' Phys] | [ '\' Logic] ';'
```

Weitere Informationen

Information über	Siehe
Statusdefinition von E/A-Netzwerken	*busstate - Status des E/A-Netzwerks auf Seite 1641*
Starten des E/A-Netzwerks	*IOBusStart - Start of I/O network auf Seite 287*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.108 IODisable – Deaktivieren eines E/A-Geräts

Verwendung

`IODisable` wird verwendet, um ein E/A-Gerät während der Programmabarbeitung zu deaktivieren.

E/A-Geräte werden nach dem Start automatisch aktiviert, sofern sie in den Systemparametern definiert sind. Bei Bedarf können E/A-Geräte während der Programmabarbeitung deaktiviert oder aktiviert werden.

Hinweis

Ein E/A-Gerät kann nicht deaktiviert werden, wenn Unit Trustlevel auf Required gesetzt ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IODisable`:

Siehe auch *Weitere Beispiele auf Seite 292*.

Beispiel 1:

```
CONST string board1:="board1";
IODisable board1, 5;
```

Deaktivieren Sie ein E/A-Gerät mit dem Namen `board1`. Warten Sie höchstens `5` Sekunden.

Argumente

```
IODisable UnitName MaxTime
```

`UnitName`

Datentyp: `string`

Ein Name eines E/A-Geräts (der Name des Geräts muss in den Systemparametern vorhanden sein)

`MaxTime`

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn die Zeit abläuft, bevor das E/A-Gerät die Deaktivierungsschritte abgeschlossen hat, wird die Fehlerbehandlung, falls vorhanden, mit dem Fehlercode `ERR_IODISABLE` aufgerufen. Wenn keine Fehlerbehandlung vorhanden ist, wird die Programmabarbeitung gestoppt. Die Deaktivierungsschritte des E/A-Geräts fahren weiter fort, unabhängig von `MaxTime` oder einem Fehler.

Das Deaktivieren eines E/A-Geräts beansprucht ca. 0-5 Sekunden.

Programmabarbeitung

Das angegebene E/A-Gerät startet die Deaktivierungsschritte. Die Instruktion ist beendet, wenn die Deaktivierungsschritte abgeschlossen sind. Wenn `MaxTime` abläuft, bevor das E/A-Gerät die Deaktivierungsschritte abgeschlossen hat, wird ein behebbarer Fehler generiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.108 IODisable – Deaktivieren eines E/A-Geräts

RobotWare - OS

Fortsetzung

Nach dem Deaktivieren eines E/A-Geräts sind alle Einstellungen für Ausgaben auf diesem Gerät fehlerhaft.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_IODISABLE	Die Wartezeit läuft ab, bevor das E/A-Gerät deaktiviert wird
ERR_NAME_INVALID	Der Name des E/A-Geräts existiert nicht.
ERR_TRUSTLEVEL	Das E/A-Gerät kann nicht deaktiviert werden, wenn Unit Trustlevel auf Required eingestellt ist.

Weitere Beispiele

Weitere Beispiele für die Instruktion IODisable sind unten aufgeführt.

Beispiel 1:

```
PROC go_home()
  VAR num recover_flag :=0;
  ...
  ! Start to deactivate I/O unit board1
  recover_flag := 1;
  IODisable "board1", 0;
  ! Move to home position
  MoveJ home, v1000,fine,tool1;
  ! Wait until deactivation of I/O unit board1 is ready
  recover_flag := 2;
  IODisable "board1", 5;
  ...
  ERROR
    IF ERRNO = ERR_IODISABLE THEN
      IF recover_flag = 1 THEN
        TRYNEXT;
      ELSEIF recover_flag = 2 THEN
        IF RemaningRetries() > 0 THEN
          RETRY;
        ELSE
          RAISE;
        ENDIF
      ENDIF
    ELSE
      ErrWrite "IODisable error", "Not possible to deactivate I/O
           unit board1";
      Stop;
    ENDIF
ENDPROC
```

Um Zykluszeit zu sparen, wird das E/A-Gerät board1 während der Roboterbewegung zu der home-Stellung deaktiviert. Mit dem Roboter in der home-Stellung wird ein Test ausgeführt, um zu ermitteln, ob das E/A-Gerät board1

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

vollständig deaktiviert ist. Nach der maximalen Anzahl von Wiederholungen (4 mit einer Wartezeit von `5` s) stoppt der Roboter mit einer Fehlermeldung.

Dasselbe Prinzip gilt für `IOEnable` (was im Vergleich zu `IODisable` mehr Zykluszeit spart).

Syntax

```
IODisable
  [ UnitName ':=' ] < expression (IN) of string> ','
  [ MaxTime ':=' ] < expression (IN) of num> ';'
```

Weitere Informationen

Information über	Siehe
Aktivieren eines E/A-Geräts	*IOEnable – Aktivieren eines E/A-Geräts auf Seite 294*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Allgemeine Ein-/Ausgang-Funktion	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.109 IOEnable – Aktivieren eines E/A-Geräts

Verwendung

`IOEnable` wird verwendet, um ein E/A-Gerät während der Programmabarbeitung zu aktivieren.

E/A-Geräte werden nach dem Start automatisch aktiviert, sofern sie in den Systemparametern definiert sind. Bei Bedarf können E/A-Geräte während der Programmabarbeitung deaktiviert oder aktiviert werden.

Die Steuerungsaktion beim Aktivieren eines E/A-Geräts hängt von den definierten Unit Trustlevel in den Systemparametern ab.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IOEnable`:

Siehe auch *Weitere Beispiele auf Seite 295*.

Beispiel 1:

```
CONST string board1:="board1";
IOEnable board1, 5;
```

Aktivieren Sie ein E/A-Gerät mit dem Namen `board1`. Warten Sie höchstens `5` Sekunden.

Argumente

```
IOEnable UnitName MaxTime
```

`UnitName`

Datentyp: `string`

Ein Name eines E/A-Geräts (der E/A-Gerätename muss in den Systemparametern vorhanden sein).

`MaxTime`

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn die Zeit abläuft, bevor das E/A-Gerät die Aktivierungsschritte abgeschlossen hat, wird die Fehlerbehandlung, falls vorhanden, mit dem Fehlercode `ERR_IOENABLE` aufgerufen. Wenn keine Fehlerbehandlung vorhanden ist, wird die Abarbeitung gestoppt. Die Aktivierungsschritte des E/A-Geräts fahren weiter fort, unabhängig von `MaxTime` oder einem Fehler.

Das Aktivieren eines E/A-Geräts beansprucht ca. 2-5 Sekunden.

Programmabarbeitung

Das angegebene E/A-Gerät startet die Aktivierungsschritte. Die Instruktion ist beendet, wenn die Aktivierungsschritte abgeschlossen sind. Wenn `MaxTime` abläuft, bevor das E/A-Gerät die Aktivierungsschritte abgeschlossen hat, wird ein behebbarer Fehler generiert.

Nach einer Sequenz von `IODisable` - `IOEnable` werden alle Ausgaben für das aktuelle E/A-Gerät auf die alten Werte gesetzt (vor `IODisable`).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_IOENABLE	Die Timeout-Zeit läuft ab, bevor das E/A-Gerät aktiviert ist.
ERR_NAME_INVALID	Der Name des E/A-Geräts existiert nicht
ERR_BUSSTATE	Das E/A-Netzwerk weist einen Fehlerzustand auf oder geht in einen Fehlerzustand über, bevor das E/A-Gerät aktiviert wird.

Weitere Beispiele

IOEnable kann auch verwendet werden, um zu prüfen, ob die Verbindung zu einem E/A-Gerät getrennt wurde.

Weitere Beispiele zur Verwendung der Instruktion IOEnable sind unten aufgeführt.

Beispiel 1:

```
VAR num max_retry:=0;
...
IOEnable "board1", 0;
SetDO board1_sig3, 1;
...
ERROR
  IF ERRNO = ERR_IOENABLE THEN
    IF RemaningRetries() > 0 THEN
      WaitTime 1;
      RETRY;
    ELSE
      RAISE;
    ENDIF
  ELSE
    ErrWrite "IOEnable error", "Not possible to activate I/O unit
         board1";
    Stop;
  ENDIF
```

Vor dem Verwenden von Signalen an board1 des E/A-Geräts erfolgt ein Test durch den Versuch, das E/A-Gerät mit einem Timeout nach 0 Sekunden zu aktivieren. Wenn der Test fehlschlägt, erfolgt ein Sprung in die Fehlerbehandlungsroutine. In der Fehlerbehandlungsroutine wartet die Programmabarbeitung 1 Sekunde lang und wiederholt dann den Versuch. Nach 4 Wiederholversuchen wird der Fehler ERR_IOENABLE an den Aufrufer dieser Routine übergeben.

Syntax

```
IOEnable
  [ UnitName ':=' ] < expression (IN) of string>' ,'
  [ MaxTime' :=' ] < expression (IN) of num > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.109 IOEnable – Aktivieren eines E/A-Geräts

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Deaktivieren eines E/A-Geräts	*IODisable – Deaktivieren eines E/A-Geräts auf Seite 291*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.110 IOEventMessage - E/A-Ereignismeldungen von Gerät ein-/ausschalten

Verwendung

`IOEventMessage` wird verwendet, um festzulegen, ob E/A-Ereignismeldungen vom E/A-Gerät gesendet werden sollen.

Dies eignet sich typischerweise für geplantes Ein-/Ausschalten des Stroms oder die Trennung des E/A-Gerätes, z. B. während eines Werkzeugwechsels mit FastDeviceStartup. Weitere Informationen zu FastDeviceStartup, siehe *Anwendungshandbuch - PROFINET Controller/Device 3HAC065546-003*.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IOEventMessage`:

Beispiel

```
PROC IOEventMessage(switch \On | switch \Off, string DeviceName);
```

Argumente

```
IOEventMessage [\On] | [\Off] DeviceName
```

`[\On]`

Datentyp: `switch`

Aktivierung der E/A-Ereignismeldungen von Gerät.

`[\Off]`

Datentyp: `switch`

Deaktivierung der E/A-Ereignismeldungen von Gerät.

Programmabarbeitung

```
PERS string TC_IO :="ToolChanger_IO"
! I/O Device name

PROC setTool(num Tool)
  !We will now switch tool and thus shut down the current active
       tool, we stop event logs while performing toolchange
  IOEventMessage \Off, TC_IO;
  ! Stop Event Messages

  IF (tool = 1) THEN
    SetDO Tool2_PS, 0;
    ! shut down Power supply Tool 2
    WaitDi Tool_Active, 0\MaxTime:=1;
    ! wait for device to shut down

    SetDO Tool1_PS, 1;
    ! power up Tool 1
    WaitDi Tool_Active, 1\MaxTime:=1;
    ! wait for device to activate

    ! Check device is up and running OK
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.110 IOEventMessage - E/A-Ereignismeldungen von Gerät ein-/ausschalten

RobotWare - OS

Fortsetzung

```
    IF (IOUnitState (TC_IO) = IOUNIT_RUNERROR) THEN
      Log "ERROR: Unit is not working because of some runtime
           error.";

    ELSEIF (IOUnitState (TC_IO) = IOUNIT_OTHERERR) THEN
    Log "ERROR: Other configuration or startup errors.";
    ENDIF

  IF (tool = 2) THEN
    !Same as above but for tool 2

  ! Toolchange finished, now we want event logs active again
  IOEventMessage \On, TC_IO;
  ! Turn Event Messages

ENDPROC
```

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_NAME_INVALID	Der Name des E/A-Geräts existiert nicht

Syntax

```
IOEventMessage
  [ '\' On] | [ '\' Off]
  [ DeviceName ' :=' ] < expression (IN) of string > ';'
```

Weitere Informationen

Information über	Siehe
Fast Device Startup (schneller Gerätestart)	*Anwendungshandbuch - PROFINET Controller/Device*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.111 IPathPos - Bei Pendelbewegug robtarget Mittellinie abrufen

Verwendung

`IPathPos` wird verwendet, um die Position der Mittellinie während der Pendelbewegung mit CAP zu erhalten.

Diese Funktion wird vor allem zusammen mit der Nachverfolgungsfunktionalität verwendet. Es ist notwendig, die Signale der Pendelbewegung und die Synchronisationssignale sowohl auf der linken Seite als auch auf der rechten Seite zu aktivieren.

Grundlegendes Beispiel

```
connect intpt, TRP_ipathpos IPathPos p_robt, sen_pos, intpt;
```

Wenn `p_robt` einen neuen berechneten Wert erhält, wird der Interrupt `intpt` gesendet, und die Trap `TRP_ipathpos` ausgeführt.

Argumente

```
IPathPos p_robt, sen_pos, intpt [\NoDispl] [\EOffs]
```

p_robt

Datentyp: `robtarget`

`p_robt` behält den aktuellen Wert des berechneten `robtarget` bei.

sen_pos

Datentyp: `pos`

`sen_pos` wird nicht verwendet.

intpt

Datentyp: `intno`

`intpt` gibt den Interrupt an, der jedes Mal empfangen wird, wenn ein neuer Wert zu `p_robt` zugeordnet wird.

[\NoDispl]

Datentyp: `switch`

Wenn `\NoDispl` angegeben ist, enthält der zurückgegebene Wert in `PERS p_robt` keine Verschiebung, die mit den RAPID-Instruktionen `PDispSet` und `PDispOn` spezifiziert werden könnte.

[\EOffs]

Datentyp: `switch`

Wenn `[\EOffs]` angegeben ist, umfasst der in `PERS p_robt` zurückgegebene Wert jeden Offset-Wert, der unter Verwendung der RAPID-Instruktion `EOffsSet` angegeben wurde.

Einschränkungen

Pendelbewegung und Synchronisation sind zu aktivieren (mit oder ohne Tracking).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.111 IPathPos - Bei Pendelbewegug robtarget Mittellinie abrufen

Continuous Application Platform (CAP)
Fortsetzung

Syntax

```
IPathPos
  [p_robt ':='] < persistent (PERS) of robtarget > ','
  [sen_pos ':='] < persistent (PERS) of pos > ','
  [Interrupt ':='] < variable (IN) of intnum >
  ['\' EOffs ]
  ['\' NoDispl ] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instruktion `CapWeaveSync`	*CapWeaveSync - Signale und Ebenen für die Synchronisierung der Pendelbewegung einrichten. auf Seite 128*
Instruktion `CapAPTrSetup`	*CapAPTrSetup - Einrichtung eines At-Point-Trackers auf Seite 83*
Instruktion `CapLATrSetup`	*CapLATrSetup - Look-Ahead-Tracker einrichten auf Seite 118*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.112 IPers - Interrupt bei Wertänderung einer persistenten Variable

Verwendung

`IPers` (*Interrupt Persistent*) wird verwendet, um Interrupts anzufordern und freizugeben, wenn der Wert einer persistenten Variable geändert wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IPers`:

Beispiel 1:

```
VAR intnum pers1int;
PERS num counter := 0;

PROC main()
  CONNECT pers1int WITH iroutine1;
  IPers counter, pers1int;
  ...
  IDelete pers1int;
ENDPROC

TRAP iroutine1
  TPWrite "Current value of counter = " \Num:=counter;
ENDTRAP
```

Fordert einen Interrupt an, der immer dann eintreten soll, wenn sich der Wert der persistenten Variable `counter` ändert. Dann erfolgt ein Aufruf an die Interrupt-Routine `iroutine1`.

Argumente

```
IPers [ \Single ] | [ \SingleSafe ] Name Interrupt
```

`[ \Single ]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll.

Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[ \SingleSafe ]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.112 IPers - Interrupt bei Wertänderung einer persistenten Variable

RobotWare - OS

Fortsetzung

`Name`

Datentyp: `anytype`

Die persistente Variable, die Interrupts generieren soll.

Es kann ein beliebiger Typ von Daten verwendet werden, z. B. Einzeldaten, Datensätze, Datensatzkomponenten, Datenfelder oder Datenfeldkomponenten.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Programmabarbeitung

Wenn sich der Wert der persistenten Variablen ändert, wird die entsprechende Interrupt-Routine aufgerufen. Nach Abschluss dieser Routine wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Falls sich der Wert der persistenten Variablen bei einem Programmstopp ändert, tritt kein Interrupt ein, wenn das Programm wieder gestartet wird.

Einschränkungen

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Siehe Instruktionen - `ISignalDI`.

Wenn im Parameter `Name` eine Einplanung für einen Teil der Daten wie z. B. eine Datensatzkomponente oder ein Datenfeldelement angegeben ist, tritt der Interrupt bei jeder Änderung dieser Daten ein.

Wenn die Interrupt-Routine ausgeführt und der Wert der Persistenten gelesen wird, ist nicht gewährleistet, dass der gelesene Wert der Wert ist, der den Interrupt ausgelöst hat.

Syntax

```
IPers
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Name ':=' ] < persistent (PERS) of anytype > ','
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt von einem analogen Eingangssignal	*ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal auf Seite 320*
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.113 IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp

Verwendung

`IRMQMessage` (*Interrupt RAPID Message Queue Message*) wird verwendet, um Interrupts für einen spezifischen Datentyp anzufordern und zu aktivieren, wenn die RMQ-Funktion verwendet wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IRMQMessage`:

Siehe auch *IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*.

Beispiel 1:

```
VAR intnum rmqint;
VAR string dummy;
...
PROC main()
  CONNECT rmqint WITH iroutine1;
  IRMQMessage dummy, rmqint;
```

Fordert einen Interrupt an, der immer dann eintreten soll, wenn eine neue `rmqmessage` mit dem Datentyp `string` empfangen wird. Anschließend wird die `iroutine1`-Routine TRAP aufgerufen.

Argumente

```
IRMQMessage InterruptDataType Interrupt
```

`InterruptDataType`

Datentyp: `anytype`

Ein Verweis auf eine Variable, Persistente oder Konstante eines Datentyps, der einen Interrupt generiert, wenn eine `rmqmessage` mit dem angegebenen Datentyp empfangen wird.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor mit einer TRAP-Routine verbunden worden sein (mithilfe der Instruktion `CONNECT`).

Programmabarbeitung

Wenn die RMQ-Meldung mit dem angegebenen Datentyp empfangen wird, erfolgt ein Aufruf der entsprechenden TRAP-Routine. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Alle Meldungen mit Daten desselben Datentyps werden unabhängig von der Anzahl der Dimensionen von demselben Interrupt behandelt. Wenn unterschiedliche Dimensionen verwendet werden, passen Sie die Abarbeitung mithilfe von `RMQGetMsgHeader` entsprechend an.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.113 IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp
FlexPendant Interface, PC Interface, or Multitasking
Fortsetzung

Jede Meldung mit Daten eines Datentyps, mit dem kein Interrupt verbunden ist, generiert eine Warnung.

Die `RMQSendWait`-Instruktion hat die höchste Priorität, wenn eine Meldung empfangen wird, die der Beschreibung der erwarteten Antwort und einer Meldung entspricht, die mithilfe der Instruktion `IRMQMessage` mit einer TRAP-Routine verbunden ist.

Im Argument `InterruptDataType` können nicht alle Datentypen verwendet werden (siehe Einschränkungen).

Der Interrupt wird als sicherer Interrupt verstanden. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `IRMQMessage` sind unten aufgeführt.

Beispiel 1:

```
MODULE ReceiverMod
  VAR intnum intno1;
  VAR rmqheader rmqheader1;
  VAR rmqslot rmqslot1;
  VAR rmqmessage rmqmessage1;

  PROC main()
    VAR string interrupt_on_str := stEmpty;
    CONNECT intno1 WITH RecMsgs;
    ! Set up interrupts for data type string
    IRMQMessage interrupt_on_str, intno1;

    ! Perform cycle
    WHILE TRUE DO
      ...
    ENDWHILE
  ENDPROC
  TRAP RecMsgs
    VAR string receivestr;
    VAR string client_name;
    VAR num userdef;

    ! Get the message from the RMQ
    RMQGetMessage rmqmessage1;
    ! Get information about the message
    RMQGetMsgHeader rmqmessage1 \Header:=rmqheader1
         \SenderId:=rmqslot1 \UserDef:=userdef;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
    IF rmqheader1.datatype = "string" AND rmqheader1.ndim = 0 THEN
      ! Get the data received in rmqmessage1
      RMQGetMsgData rmqmessage1, receivestr;
      client_name := RMQGetSlotName(rmqslot1);
      TPWrite "Rec string: " + receivestr;
      TPWrite "User Def: " + ValToStr(userdef);
      TPWrite "From: " + client_name;
    ELSE
      TPWrite "Faulty data received!"
    ENDIF

  ENDTRAP
ENDMODULE
```

In dem Beispiel wird gezeigt, wie Interrupts für einen bestimmten Datentyp eingerichtet werden. Beim Empfang einer Meldung wird `TRAP RecMsgs` ausgeführt und die Daten in der empfangenen Meldung werden auf dem FlexPendant ausgegeben. Wenn der empfangene Datentyp oder die Dimension der Daten nicht mit dem erwarteten Datentyp bzw. der erwarteten Dimension übereinstimmt, wird dies auf dem FlexPendant ausgegeben.

Einschränkungen

Es ist nicht erlaubt, `IRMQMessage` im Synchron-Modus abzuarbeiten. Dies würde zu einem schwerwiegenden Laufzeitfehler führen.

Das Festlegen von Interrupts und das Senden oder Empfangen von Dateninstanzen von Datentypen mit keinem Wert (Non-Value), festem Wert (Semi-Value) oder vom Datentyp `motsetdata.` ist nicht möglich.

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
VAR intnum rmqint;
PROC main ()
  VAR mytype dummy;
  CONNECT rmq1int WITH iroutine1;
  IRMQMessage dummy, rmqint;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Sämtliche Aktivierungen von Interrupts erfolgen am Beginn des Programms. Diese Instruktionen werden dann außerhalb des Hauptablaufs des Programms gehalten.

```
VAR intnum rmqint;
PROC main ( )
  VAR mytype dummy;
  CONNECT rmqint WITH iroutine1;
  IRMQMessage dummy, rmqint;
  ...
  IDelete rmqint;
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.113 IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

Der Interrupt wird am Ende des Programms gelöscht und dann erneut aktiviert. Beachten Sie in diesem Fall, dass der Interrupt für eine kurze Zeit inaktiv ist.

Syntax

```
IRMQMessage
  [ InterruptDataType ':=' ] < reference (REF) of anytype >
  [ Interrupt ':=' ] < variable (VAR) of intnum >';'
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*
Senden von Daten an die Warteschlange einer RAPID-Task oder eines Robot Application Builder-Clients	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange einer RAPID-Task oder eines Robot Application Builder-Clients und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Headerdaten aus einer `rmqmessage`.	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Senden von Daten an die Warteschlange einer RAPID-Task oder eines Robot Application Builder-Clients	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Extrahieren der Daten aus einer `rmqmessage`.	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität.	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.114 ISignalAI - Interrupts von analogem Eingangssignal

Verwendung

`ISignalAI` (*Interrupt Signal Analog Input*) wird verwendet, um Interrupts von einem analogen Eingangssignal anzufordern und freizugeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ISignalAI`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalAI \Single, ai1, AIO_BETWEEN, 1.5, 0.5, 0, sig1int;
```

Fordert einen Interrupt an, wenn der logische Wert des analogen Eingangssignals `ai1` zwischen `0.5` und `1.5` liegt. Dann erfolgt ein Aufruf der Interrupt-Routine `iroutine1`.

Beispiel 2:

```
ISignalAI ai1, AIO_BETWEEN, 1.5, 0.5, 0.1, sig1int;
```

Fordert jedes Mal einen Interrupt an, wenn der logische Wert des analogen Eingangssignals `ai1` zwischen `0.5` und `1.5` liegt und die absolute Signaldifferenz im Vergleich zum vorherigen gespeicherten Referenzwert größer als `0.1` ist.

Beispiel 3

```
ISignalAI ai1, AIO_OUTSIDE, 1.5, 0.5, 0.1, sig1int;
```

Fordert jedes Mal einen Interrupt an, wenn der logische Wert des analogen Eingangssignals `ai1` kleiner als 0,5 oder größer als 1,5 ist und die absolute Signaldifferenz im Vergleich zum vorherigen gespeicherten Referenzwert größer als 0,1 ist.

Argumente

```
ISignalAI [\Single] | [\SingleSafe] Signal Condition HighValue
     LowValue DeltaValue [\DPos] | [\DNeg] Interrupt
```

`[\Single]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll. Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[\SingleSafe]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

`Signal`

Datentyp: `signalai`

Der Name des Signals, das Interrupts generieren soll.

`Condition`

Datentyp: `aiotrigg`

Gibt an, wie `HighValue` und `LowValue` die zu erfüllende Bedingung definieren:

Wert	Symbolische Konstante	Kommentar
1	`AIO_ABOVE_HIGH`	Signal erzeugt Interrupts, wenn angegebener oberer Wert überschritten wird.
2	`AIO_BELOW_HIGH`	Signal erzeugt Interrupts, wenn angegebener oberer Wert unterschritten wird.
3	`AIO_ABOVE_LOW`	Signal erzeugt Interrupts, wenn angegebener unterer Wert überschritten wird.
4	`AIO_BELOW_LOW`	Signal erzeugt Interrupts, wenn angegebener unterer Wert unterschritten wird.
5	`AIO_BETWEEN`	Signal erzeugt Interrupts, wenn es zwischen dem angegebenen unteren und oberen Wert liegt.
6	`AIO_OUTSIDE`	Signal erzeugt Interrupts, wenn angegebener unterer Wert unterschritten oder oberer Wert überschritten wird.
7	`AIO_ALWAYS`	Signal erzeugt immer Interrupts.

`HighValue`

Datentyp: `num`

Hoher logischer Wert für die Definition der Bedingung.

`LowValue`

Datentyp: `num`

Niedriger logischer Wert für die Definition der Bedingung.

`DeltaValue`

Datentyp: `num`

Definiert die minimale logische Signaldifferenz vor der Generierung eines neuen Interrupts. Der aktuelle Signalwert muss im Vergleich zum gespeicherten Referenzwert größer als der angegebene `DeltaValue` sein, bevor ein neuer Interrupt generiert wird.

`[\DPos]`

Datentyp: `switch`

Gibt an, dass nur positive logische Signaldifferenzen neue Interrupts ergeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\DNeg]`

Datentyp: `switch`

Gibt an, dass nur negative logische Signaldifferenzen neue Interrupts ergeben.

Wenn keines der Argumente `\DPos` und `\DNeg` verwendet wird, erzeugen sowohl positive als auch negative Differenzen neue Interrupts.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Dieser Interrupt sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Programmabarbeitung

Wenn das Signal die angegebenen Bedingungen erfüllt (sowohl `Condition` als auch `DeltaValue`), wird die entsprechende Interrupt-Routine aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Bedingungen für die Interrupt-Erzeugung

Vor dem Einplanen des Interrupts wird bei jedem Ablesen des Signals der Wert des Signals gelesen, gespeichert und später als Referenzwert für die Bedingung `DeltaValue` verwendet.

Bei der Einplanung des Interrupts (wenn der angegebene `DeltaValue` = 0 ist) und nach der Einplanung des Interrupts wird das Signal abgelesen. Anschließend wird der Signalwert entsprechend `Condition` und unter Berücksichtigung von `DeltaValue` mit `HighValue` und `LowValue` verglichen, um zu bestimmen, ob ein Interrupt generiert werden soll. Wenn der neue abgelesene Wert die angegebene `HighValue` und `LowValueCondition` erfüllt, die Signaldifferenz jedoch kleiner oder gleich dem Argument `DeltaValue` ist, erfolgt kein Interrupt. Wenn die Signaldifferenz nicht das angegebene Vorzeichen aufweist, erfolgen keine Interrupts (Argument `\DPos` oder `\DNeg`).

Der gespeicherte Referenzwert für die Bedingung `DeltaValue` wird mit einem neu abgelesenen Wert für die spätere Verwendung aktualisiert, wenn die folgenden Bedingungen erfüllt sind:

- Argument `Condition` mit angegebenem `HighValue` und `LowValue` (innerhalb der Grenzen)
- Argument `DeltaValue`(ausreichende absolute Signaländerung, unabhängig von den angegebenen Argumenten `\DPos` oder `\DNeg`)

Der Referenzwert wird nur beim Ablesen aktualisiert, nicht beim Auslösen des Interrupts.

Ein Interrupt wird auch erzeugt, wenn bei der Aktualisierung des Referenzwerts das Vorzeichen der Signaldifferenz mit dem angegebenen Argument (beliebiges Vorzeichen, `\DPos0` oder `\DNeg`) übereinstimmt.

Wenn der Schalter `\Single` gesetzt ist, wird höchstens ein Interrupt generiert. Wenn der Schalter `\Single` (zyklischer Interrupt) nicht verwendet wird, erfolgt bei jedem Ablesen des Signalwerts eine erneute Überprüfung der angegebenen

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

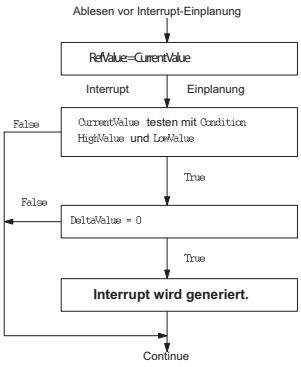
1.114 ISignalAI - Interrupts von analogem Eingangssignal

RobotWare - OS

Fortsetzung

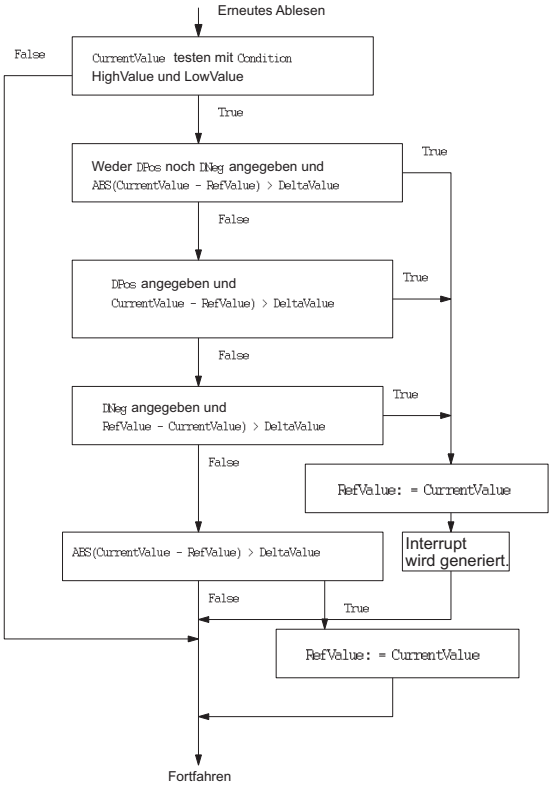
Bedingungen (sowohl `Condition`als auch `DeltaValue`). Der aktuelle Signalwert und der zuletzt gespeicherte Referenzwert werden verglichen, um zu bestimmen, ob ein Interrupt generiert werden soll.

Bedingung für Interrupt-Generierung bei der Interrupt-Einplanung



xx0500002165

Bedingung für Interrupt-Generierung bei jedem Ablesen nach der Interrupt-Einplanung

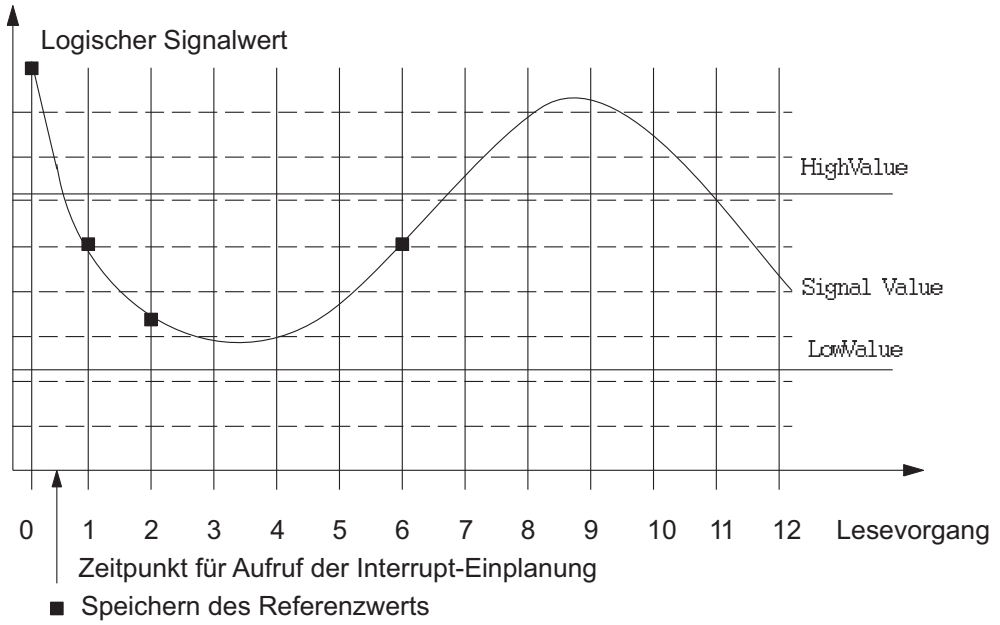


xx0500002166

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 1 für Interrupt-Generierung



xx0500002167

Angenommen, der Interrupt wird zwischen Lesevorgang 0 und 1 aufgerufen, führt nachstehende Instruktion zu folgendem Ergebnis:

```
ISignalAI ai1, AIO_BETWEEN, 6.1, 2.2, 1.0, sig1int;
```

Lesevorgang 1 generiert einen Interrupt, da der Signalwert zwischen `HighValue` **und** `LowValue` **liegt und die Signaldifferenz verglichen mit Lesevorgang 0 größer als** `DeltaValue` **ist.**

Lesevorgang 2 generiert einen Interrupt, da der Signalwert zwischen `HighValue` **und** `LowValue` **liegt und die Signaldifferenz verglichen mit Lesevorgang 1 größer als** `DeltaValue` **ist.**

Die Lesevorgänge 3, 4, 5 generieren keinen Interrupt, da die Signaldifferenz kleiner als `DeltaValue` **ist.**

Lesevorgang 6 generiert einen Interrupt.

Lesevorgänge 7 und 10 generieren keinen Interrupt, da der Signalwert über `HighValue` **liegt.**

Lesevorgang 11 generiert keinen Interrupt, da die Signaldifferenz verglichen mit Lesevorgang 6 gleich `DeltaValue` **ist.**

Lesevorgang 12 generiert keinen Interrupt, da die Signaldifferenz verglichen mit Lesevorgang 6 kleiner als `DeltaValue` **ist.**

Fortsetzung auf nächster Seite

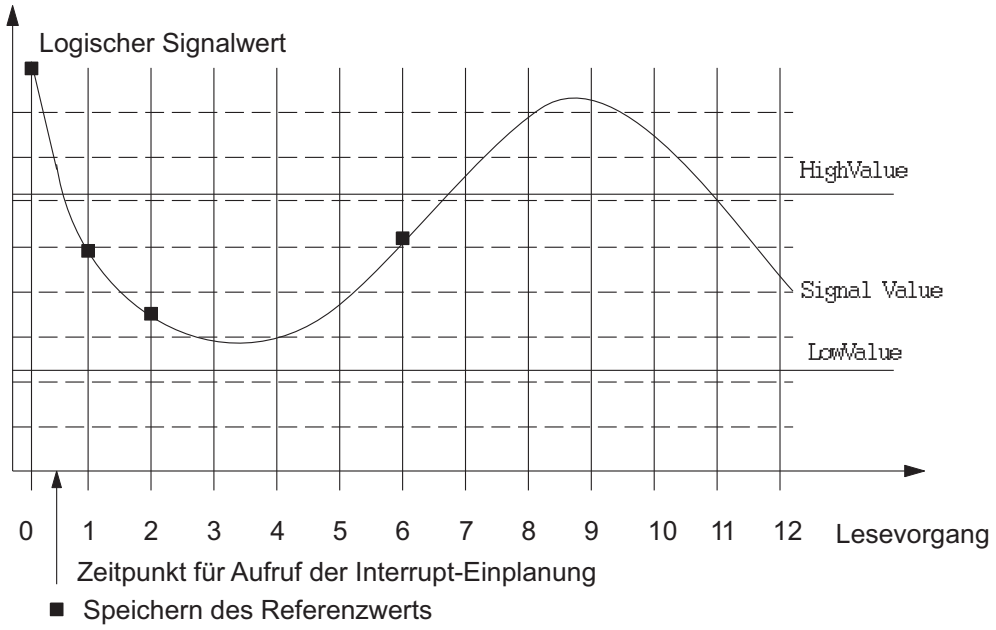
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.114 ISignalAI - Interrupts von analogem Eingangssignal

RobotWare - OS

Fortsetzung

Beispiel 2 für Interrupt-Generierung



xx0500002168

Angenommen, der Interrupt wird zwischen Lesevorgang 0 und 1 aufgerufen, führt nachstehende Instruktion zu folgendem Ergebnis:

```
ISignalAI ai1, AIO_BETWEEN, 6.1, 2.2, 1.0 \DPos, sig1int;
```

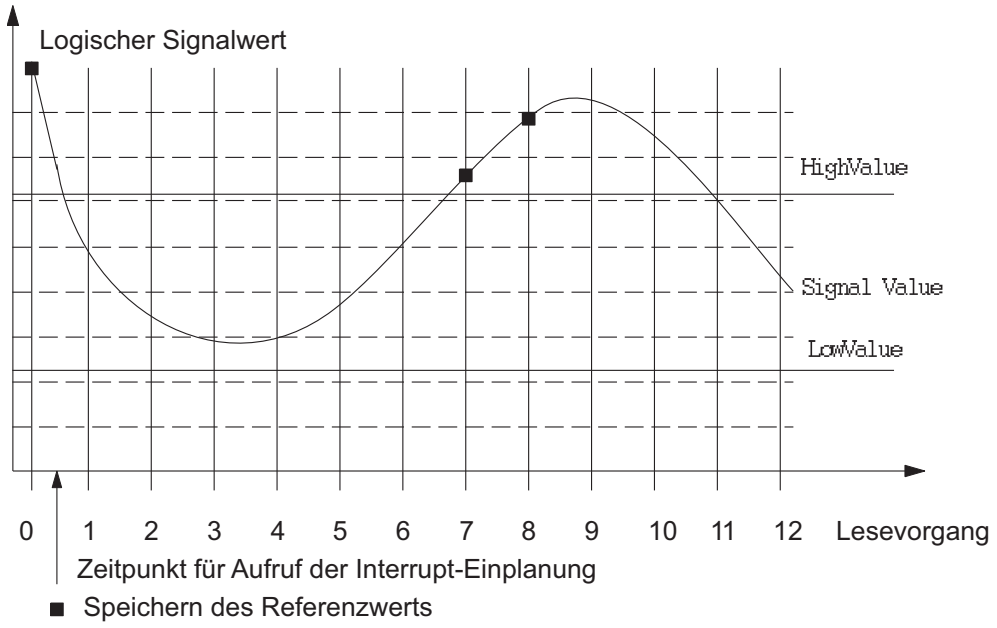
Bei Lesevorgang 1 und 2 wird ein neuer Referenzwert gespeichert, da das Signal innerhalb der Grenzen liegt und die absolute Signaldifferenz zwischen dem aktuellen Wert und dem zuletzt gespeicherten Referenzwert größer als 1,0 ist. Kein Interrupt wird generiert, da die Signaländerungen in negativer Richtung erfolgen.

Lesevorgang 6 generiert einen Interrupt, da der Signalwert zwischen `HighValue` und `LowValue` liegt und die Signaldifferenz in positiver Richtung verglichen mit Lesevorgang 2 größer als `DeltaValue` ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 3 für Interrupt-Generierung



xx0500002169

Angenommen, der Interrupt wird zwischen Lesevorgang 0 und 1 aufgerufen, führt nachstehende Instruktion zu folgendem Ergebnis:

```
ISignalAI \Single, ai1, AIO_OUTSIDE, 6.1, 2.2, 1.0 \DPos, sig1int;
```

Bei Lesevorgang 7 wird ein neuer Referenzwert gespeichert, da das Signal innerhalb der Grenzen liegt und die absolute Signaldifferenz zwischen dem aktuellen Wert und dem zuletzt gespeicherten Referenzwert größer als 1,0 ist

Lesevorgang 8 generiert einen Interrupt, da der Signalwert über `HighValue` liegt und die Signaldifferenz in positiver Richtung verglichen mit Lesevorgang 7 größer als `DeltaValue` ist.

Fortsetzung auf nächster Seite

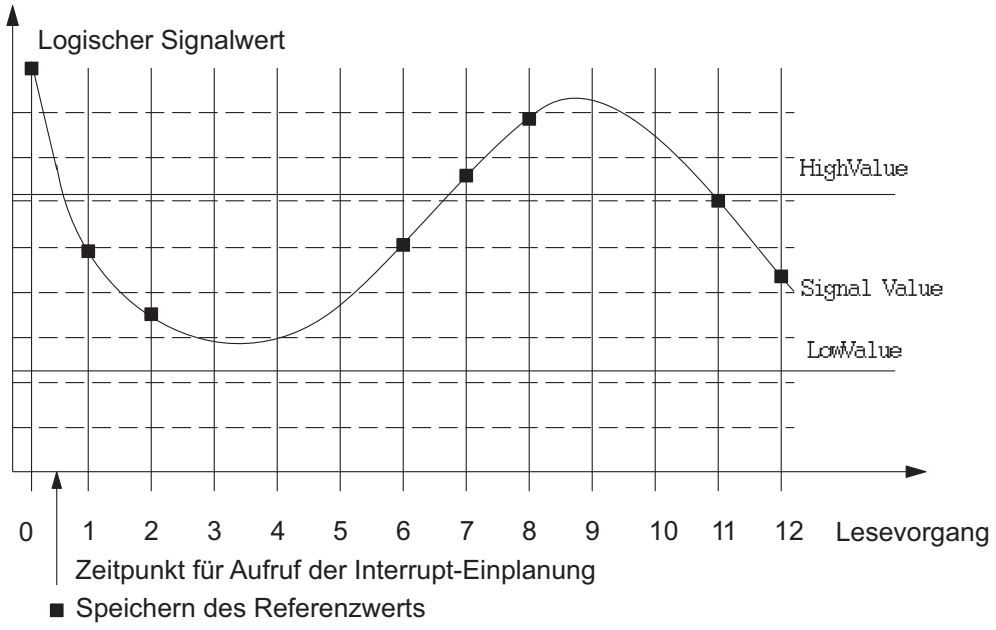
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.114 ISignalAI - Interrupts von analogem Eingangssignal

RobotWare - OS

Fortsetzung

Beispiel 4 für Interrupt-Generierung



xx0500002170

Angenommen, der Interrupt wird zwischen Lesevorgang 0 und 1 aufgerufen, führt nachstehende Instruktion zu folgendem Ergebnis:

```
ISignalAI ai1, AIO_ALWAYS, 6.1, 2.2, 1.0 \DPos, sig1int;
```

Bei Lesevorgang 1 und 2 wird ein neuer Referenzwert gespeichert, da das Signal innerhalb der Grenzen liegt und die absolute Signaldifferenz zwischen dem aktuellen Wert und dem zuletzt gespeicherten Referenzwert größer als 1,0 ist

Lesevorgang 6 generiert einen Interrupt, da die Signaldifferenz in positiver Richtung verglichen mit Lesevorgang 2 größer als `DeltaValue` ist.

Die Lesevorgänge 7 und 8 generieren einen Interrupt, da die Signaldifferenz in positiver Richtung verglichen mit dem vorherigen Lesevorgang größer als `DeltaValue` ist.

Bei Lesevorgang 11 und 12 wird ein neuer Referenzwert gespeichert, da das Signal innerhalb der Grenzen liegt und die absolute Signaldifferenz zwischen dem aktuellen Wert und dem zuletzt gespeicherten Referenzwert größer als 1,0 ist

Fehlerbehandlung

Wenn ein Interrupt für ein analoges Eingangssignal eingeplant wird, wird jedes Mal ein Interrupt erzeugt, wenn die bei der Einplanung angegebenen Bedingungen erfüllt sind. Wenn das analoge Eingangssignal verrauscht ist, können viele Interrupts generiert werden, selbst wenn sich der analoge Wert nur um ein oder zwei Bits ändert.

Um das Generieren von Interrupts für kleine Änderungen des analogen Eingangswertes zu verhindern, setzen Sie `DeltaValue` auf einen Wert größer als 0. Dann werden erst Interrupts generiert, wenn eine Änderung des analoges Werts größer als der angegebene `DeltaValue` ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_AO_LIM	Das programmierte HighValue- oder LowValue-Argument für das angegebene analoge Eingangssignal Signal liegt außerhalb des zulässigen Bereichs.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Die Argumente HighValue und LowValue sollten im folgenden Bereich liegen: zwischen logischem Höchstwert und logischem Mindestwert, die für das Signal definiert sind.

HighValue muss größer als LowValue sein.

DeltaValue muss 0 oder positiv sein.

Für die Interrupt-Identität gelten dieselben Einschränkungen wie für ISignalDI.

Syntax

```
ISignalAI
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Signal ':=' ] <variable (VAR) of signalai> ','
  [ Condition ':=' ] <expression (IN) of aiotrigg> ','
  [ HighValue ':=' ] <expression (IN) of num> ','
  [ LowValue ':=' ] <expression (IN) of num> ','
  [ DeltaValue ':=' ] <expression (IN) of num>
  [['\'DPos] | [ '\' DNeg] ',']
  [ Interrupt ':=' ] <variable (VAR) of intnum> ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Definition von Konstanten	*aiotrigg - Bedingung für analogen E/A-Trigger auf Seite 1635*
Interrupt von analogem Ausgangssignal	*ISignalAO - Interrupts von analogem Ausgangssignal auf Seite 316*
Interrupt von einem digitalen Eingangssignal	*ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal auf Seite 320*
Interrupt von einem digitalen Ausgangssignal	*ISignalDO - Interrupts von einem digitalen Ausgangssignal auf Seite 324*
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*
Verwandte Systemparameter (Filter)	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.115 ISignalAO - Interrupts von analogem Ausgangssignal

Verwendung

`ISignalAO` (*Interrupt Signal Analog Output*) wird verwendet, um Interrupts von einem analogen Ausgangssignal anzufordern und freizugeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ISignalAO`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalAO \Single, ao1, AIO_BETWEEN, 1.5, 0.5, 0, sig1int;
```

Fordert einen Interrupt an, wenn der logische Wert des analogen Ausgangssignals `ao1` erstmals zwischen `0.5` und `1.5` liegt. Dann erfolgt ein Aufruf der Interrupt-Routine `iroutine1`.

Beispiel 2:

```
ISignalAO ao1, AIO_BETWEEN, 1.5, 0.5, 0.1, sig1int;
```

Fordert jedes Mal einen Interrupt an, wenn der logische Wert des analogen Eingangssignals `ao1` zwischen `0.5` und `1.5` liegt und die absolute Signaldifferenz im Vergleich zum vorherigen gespeicherten Referenzwert größer als 0,1 ist.

Beispiel 3

```
ISignalAO ao1, AIO_OUTSIDE, 1.5, 0.5, 0.1, sig1int;
```

Fordert jedes Mal einen Interrupt an, wenn der logische Wert des analogen Ausgangssignals `ao1` kleiner als `0.5` oder größer als `1.5` ist und die absolute Signaldifferenz im Vergleich zum vorherigen gespeicherten Referenzwert größer als 0,1 ist.

Argumente

```
ISignalAO [\Single] | [\SingleSafe] Signal Condition HighValue
     LowValue DeltaValue [\DPos] | [\DNeg] Interrupt
```

`[\Single]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll. Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[\SingleSafe]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

`Signal`

Datentyp: `signalao`

Der Name des Signals, das Interrupts generieren soll.

`Condition`

Datentyp: `aiotrigg`

Gibt an, wie `HighValue` und `LowValue` die zu erfüllende Bedingung definieren:

Wert	Symbolische Konstante	Kommentar
1	`AIO_ABOVE_HIGH`	Signal erzeugt Interrupts, wenn angegebener oberer Wert überschritten wird.
2	`AIO_BELOW_HIGH`	Signal erzeugt Interrupts, wenn angegebener oberer Wert unterschritten wird.
3	`AIO_ABOVE_LOW`	Signal erzeugt Interrupts, wenn angegebener unterer Wert überschritten wird.
4	`AIO_BELOW_LOW`	Signal erzeugt Interrupts, wenn angegebener unterer Wert unterschritten wird.
5	`AIO_BETWEEN`	Signal erzeugt Interrupts, wenn es zwischen dem angegebenen unteren und oberen Wert liegt.
6	`AIO_OUTSIDE`	Signal erzeugt Interrupts, wenn angegebener unterer Wert unterschritten oder oberer Wert überschritten wird.
7	`AIO_ALWAYS`	Signal erzeugt immer Interrupts.

`HighValue`

Datentyp: `num`

Hoher logischer Wert für die Definition der Bedingung.

`LowValue`

Datentyp: `num`

Niedriger logischer Wert für die Definition der Bedingung.

`DeltaValue`

Datentyp: `num`

Definiert die minimale logische Signaldifferenz vor der Generierung eines neuen Interrupts. Der aktuelle Signalwert muss im Vergleich zum vorherigen gespeicherten Referenzwert größer als der angegebene `DeltaValue` sein, bevor ein neuer Interrupt generiert wird.

`[\DPos]`

Datentyp: `switch`

Gibt an, dass nur positive logische Signaldifferenzen neue Interrupts ergeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.115 ISignalAO - Interrupts von analogem Ausgangssignal

RobotWare - OS

Fortsetzung

`[\DNeg]`

Datentyp: `switch`

Gibt an, dass nur negative logische Signaldifferenzen neue Interrupts ergeben.

Wenn weder das Argument `\DPos` noch `\DNeg` verwendet wird, erzeugen sowohl positive als auch negative Differenzen neue Interrupts.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Dieser Interrupt sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Programmabarbeitung

Weitere Informationen über die Instruktion `ISignalAI` finden Sie unter:

- Programmabarbeitung
- Bedingung für Interrupt-Erzeugung
- Weitere Beispiele

Einige Prinzipien gelten für `ISignalAO` und für `ISignalAI`.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_AO_LIM`	Das programmierte `HighValue`- oder `LowValue`-Argument für das angegebene analoge Eingangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Die Argumente `HighValue` und `LowValue` sollten im folgenden Bereich liegen: zwischen dem logischen Höchstwert und dem logischen Mindestwert, der für das Signal definiert ist.

`HighValue` muss größer als `LowValue` sein.

`DeltaValue` muss 0 oder positiv sein.

Für die Interrupt-Identität gelten dieselben Einschränkungen wie für `ISignalDO`.

Syntax

```
ISignalAO
[ '\' Single ] | [ '\' SingleSafe ] ','
[ Signal':=' ]<variable (VAR) of signalao>',’
[ Condition':=' ]<expression (IN) of aiotrigg>','
[ HighValue':=' ]<expression (IN) of num>','
[ LowValue':=' ]<expression (IN) of num>','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[ DeltaValue':=' ]<expression (IN) of num>
[['\'DPos] | [ '\'DNeg] ','']
[ Interrupt':=' ]<variable (VAR) of intnum>';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Definition von Konstanten	*aiotrigg - Bedingung für analogen E/A-Trigger auf Seite 1635*
Interrupt von analogem Eingangssignal	*ISignalAI - Interrupts von analogem Eingangssignal auf Seite 307*
Interrupt von einem digitalen Eingangssignal	*ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal auf Seite 320*
Interrupt von einem digitalen Ausgangssignal	*ISignalDO - Interrupts von einem digitalen Ausgangssignal auf Seite 324*
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*
Verwandte Systemparameter (Filter)	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.116 ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal

Verwendung

`ISignalDI` (*Interrupt Signal Digital In*) wird verwendet, um Interrupts von einem digitalen Eingangssignal anzufordern und freizugeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ISignalDI`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1,1,sig1int;
```

Fordert einen Interrupt an, der jedes Mal auftreten soll, wenn das digitale Eingangssignal `di1` auf `1` gesetzt wird. Dann erfolgt ein Aufruf an die Interrupt-Routine `iroutine1`.

Beispiel 2:

```
ISignalDI di1,0,sig1int;
```

Fordert einen Interrupt an, der jedes Mal auftreten soll, wenn das digitale Eingangssignal `di1` auf `0` gesetzt wird.

Beispiel 3

```
ISignalDI \Single, di1,1,sig1int;
```

Fordert einen Interrupt an, der nur das erste Mal auftreten soll, wenn das digitale Eingangssignal `di1` auf `1` gesetzt wird.

Argumente

```
ISignalDI [ \Single] | [ \SingleSafe] Signal TriggValue Interrupt
```

`[ \Single ]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll.

Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[ \SingleSafe ]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Signal`

Datentyp: `signaldi`

Der Name des Signals, das Interrupts generieren soll.

`TriggValue`

Datentyp: `dionum`

Der Wert, auf den das Signal wechseln muss, damit ein Interrupt eintritt.

Der Wert wird als 0 oder 1 oder als symbolischer Wert (z. B. `high/low`) angegeben. Das Signal wird beim Wechsel auf 0 oder 1 durch die steigende oder fallende Flanke ausgelöst.

`TriggValue` 2 oder der symbolische Wert `edge` kann zur Erzeugung von Interrupts sowohl auf der positiven Flanke (0 -> 1) als auch auf der negativen Flanke (1 -> 0) verwendet werden.

`Interrupt`

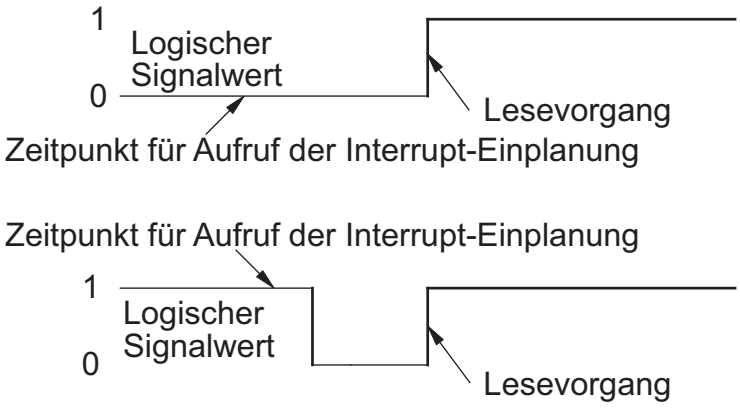
Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Programmabarbeitung

Wenn das Signal den angegebenen Wert annimmt, wird die entsprechende Interrupt-Routine aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Wenn sich das Signal in den angegebenen Wert ändert, bevor der Interrupt angefordert wird, tritt kein Interrupt ein. In der Abbildung unten sind Interrupts von einem digitalen Eingangssignal bei Signalebene 1 abgebildet.



xx0500002189

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.116 ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1, 1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Sämtliche Aktivierungen von Interrupts erfolgen am Beginn des Programms. Diese Instruktionen werden dann außerhalb des Hauptablaufs des Programms gehalten.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1, 1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Der Interrupt wird am Ende des Programms gelöscht und dann erneut aktiviert. Beachten Sie in diesem Fall, dass der Interrupt für eine kurze Zeit inaktiv ist.

Syntax

```
ISignalDI
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Signal ':=' ] < variable (VAR) of signaldi > ','
  [ TriggValue' :=' ] < expression (IN) of dionum > ','
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt von einem analogen Ausgangssignal	*ISignalDO - Interrupts von einem digitalen Ausgangssignal auf Seite 324*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.116 ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal

RobotWare - OS

Fortsetzung

Information über	Siehe
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.117 ISignalDO - Interrupts von einem digitalen Ausgangssignal

Verwendung

`ISignalDO` (*Interrupt Signal Digital Out*) wird verwendet, um Interrupts von einem digitalen Eingangssignal anzufordern und freizugeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ISignalDO`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1,1,sig1int;
```

Fordert einen Interrupt an, der jedes Mal auftreten soll, wenn das digitale Ausgangssignal `do1` auf `1` gesetzt wird. Dann erfolgt ein Aufruf an die Interrupt-Routine `iroutine1`.

Beispiel 2:

```
ISignalDO do1,0,sig1int;
```

Fordert einen Interrupt an, der jedes Mal auftreten soll, wenn das digitale Ausgangssignal `do1` auf `0` gesetzt wird.

Beispiel 3

```
ISignalDO\Single, do1,1,sig1int;
```

Fordert einen Interrupt an, der nur das erste Mal auftreten soll, wenn das digitale Ausgangssignal `do1` auf `1` gesetzt wird.

Argumente

```
ISignalDO [ \Single ] | [ \SingleSafe ] Signal TriggValue Interrupt
```

`[ \Single ]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll.

Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[ \SingleSafe ]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Signal`

Datentyp: `signaldo`

Der Name des Signals, das Interrupts generieren soll.

`TriggValue`

Datentyp: `dionum`

Der Wert, auf den das Signal wechseln muss, damit ein Interrupt eintritt.

Der Wert wird als 0 oder 1 oder als symbolischer Wert (z. B. `high/low`) angegeben. Das Signal wird beim Wechsel auf 0 oder 1 durch die steigende oder fallende Flanke ausgelöst.

`TriggValue` 2 oder der symbolische Wert `edge` kann zur Erzeugung von Interrupts sowohl auf der positiven Flanke (0 -> 1) als auch auf der negativen Flanke (1 -> 0) verwendet werden.

`Interrupt`

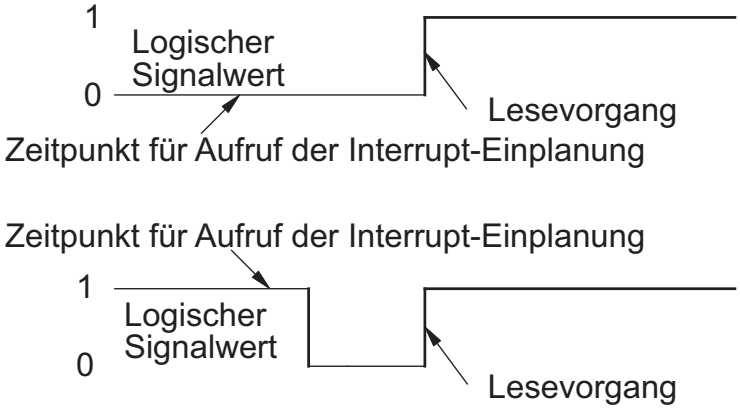
Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor mit einer Interrupt-Routine verbunden worden sein (mithilfe der Instruktion `CONNECT`.

Programmabarbeitung

Wenn das Signal den angegebenen Wert 0 oder 1 annimmt, wird die entsprechende Interrupt-Routine aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Wenn sich das Signal in den angegebenen Wert ändert, bevor der Interrupt angefordert wird, tritt kein Interrupt ein. In der Abbildung unten sind Interrupts von einem digitalen Ausgangssignal bei Signalebene 1 abgebildet.



xx0500002190

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.117 ISignalDO - Interrupts von einem digitalen Ausgangssignal
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1, 1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Sämtliche Aktivierungen von Interrupts erfolgen am Beginn des Programms. Diese Instruktionen werden dann außerhalb des Hauptablaufs des Programms gehalten.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalDO do1, 1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Der Interrupt wird am Ende des Programms gelöscht und dann erneut aktiviert. Beachten Sie in diesem Fall, dass der Interrupt für eine kurze Zeit inaktiv ist.

Syntax

```
ISignalDO
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Signal ':=' ] < variable (VAR) of signaldo > ','
  [ TriggValue' :=' ] < expression (IN) of dionum > ','
  [ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt von einem analogen Eingangssignal	*ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal auf Seite 320*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.117 ISignalDO - Interrupts von einem digitalen Ausgangssignal

RobotWare - OS

Fortsetzung

Information über	Siehe
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.118 ISignalGI - Anfordern von Interrupts von einer Gruppe digitaler Eingangssignale

Verwendung

`ISignalGI` (*Interrupt Signal Group Digital In*) wird verwendet, um Interrupts von einer Gruppe digitaler Eingangssignale anzufordern und freizugeben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ISignalGI`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalGI gi1,sig1int;
```

Fordert einen Interrupt an, wenn der Wert eines digitalen Gruppeneingangssignals geändert wird.

Argumente

```
ISignalGI [ \Single ] | [ \SingleSafe ] Signal Interrupt
```

`[ \Single ]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll.

Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[ \SingleSafe ]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

`Signal`

Datentyp: `signalgi`

Der Name des Gruppeneingangssignals, das Interrupts generiert.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Wenn sich der Wert des Gruppensignals ändert, wird die entsprechende Interrupt-Routine aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Wenn sich das Signal ändert, bevor der Interrupt angefordert wird, tritt kein Interrupt ein.

Wenn ein digitales Gruppeneingangssignal auf einen Wert gesetzt wird, kann es verschiedene Interrupts generieren. Der Grund dafür ist, dass Änderungen an den einzelnen Bits im Gruppensignal nicht gleichzeitig vom Robotersystem erkannt werden. Um mehrere Interrupts für eine Gruppensignaländerung zu vermeiden, kann für das Signal eine Filterzeit definiert werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Die maximale Anzahl an Signalen, die für eine Gruppe verwendet werden kann, beträgt 32.

Eine numerische Wert-Bedingung kann nicht in der Instruktion verwendet werden, um festzulegen, dass ein Interrupt aufgerufen werden soll, wenn dieser bestimmte Wert geändert wird. Dies muss über das Benutzerprogramm geschehen, indem der Wert des Gruppensignals bei der Abarbeitung der TRAP-Routine gelesen wird.

Die Interrupts werden als Bit-Interrupts generiert, d. h. als Interrupts bei Änderungen einzelner digitaler Eingangssignale. Wenn sich die Bits im Gruppensignal mit einer Verzögerung zwischen den Einstellungen ändern, werden mehrere Interrupts generiert. Kenntnisse über die Funktionsweise der E/A-Karte sind wichtig, um bei Verwendung von ISignalGI die richtige Funktionalität zu erhalten. Wenn bei den Gruppeneingangseinstellungen mehrere Interrupts generiert werden, verwenden sie stattdessen ISignalDI für ein Impulssignal, das festgelegt wird, wenn alle Bits im Gruppensignal festgelegt wurden.

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGI gi1, sig1int;
  WHILE TRUE DO
  ...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.118 ISignalGI - Anfordern von Interrupts von einer Gruppe digitaler Eingangssignale

RobotWare - OS

Fortsetzung

```
  ENDWHILE
ENDPROC
```

Sämtliche Aktivierungen von Interrupts erfolgen am Beginn des Programms. Diese Instruktionen werden dann außerhalb des Hauptablaufs des Programms gehalten.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGI gi1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Der Interrupt wird am Ende des Programms gelöscht und dann erneut aktiviert. Beachten Sie in diesem Fall, dass der Interrupt für eine kurze Zeit inaktiv ist.

Syntax

```
ISignalGI
  ['\' Single] | ['\' SingleSafe]','
  [Signal ':=' ] < variable (VAR) of signalgi>','
  [Interrupt ':='] <variable (VAR) of intnum>';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt von einem analogen Eingangssignal	*ISignalDI - Anfordern von Interrupts von einem digitalen Eingangssignal auf Seite 320*
Interrupt von Gruppenausgangssignalen	*ISignalGO - Anfordern von Interrupts von einer Gruppe digitaler Ausgangssignale auf Seite 331*
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*
Filterzeit	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.119 ISignalGO - Anfordern von Interrupts von einer Gruppe digitaler Ausgangssignale

Verwendung

`ISignalGO` (*Interrupt Signal Group Digital Out*) wird verwendet, um Interrupts von einer Gruppe digitaler Ausgangssignale anzufordern und freizugeben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ISignalGO`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalGO go1,sig1int;
```

Fordert einen Interrupt an, wenn der Wert eines digitalen Gruppenausgangssignals geändert wird.

Argumente

```
ISignalGO [ \Single ] | [ \SingleSafe ] Signal Interrupt
```

`[ \Single ]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll.

Wenn das Argument `Single` gesetzt ist, tritt der Interrupt höchstens einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal auf, wenn seine Bedingung erfüllt ist.

`[ \SingleSafe ]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

`Signal`

Datentyp: `signalgo`

Der Name des Gruppenausgangssignals, das Interrupts generiert.

`Interrupt`

Datentyp: `intnum`

Die Interrupt-Identität. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.119 ISignalGO - Anfordern von Interrupts von einer Gruppe digitaler Ausgangssignale
RobotWare - OS
Fortsetzung

Programmabarbeitung

Wenn sich der Wert des Gruppensignals ändert, wird die entsprechende Interrupt-Routine aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Wenn sich das Signal ändert, bevor der Interrupt angefordert wird, tritt kein Interrupt ein.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Die maximale Anzahl an Signalen, die für eine Gruppe verwendet werden kann, beträgt 32.

Eine numerische Wert-Bedingung kann nicht in der Instruktion verwendet werden, um festzulegen, dass ein Interrupt aufgerufen werden soll, wenn dieser bestimmte Wert geändert wird. Dies muss über das Benutzerprogramm geschehen, indem der Wert des Gruppensignals bei der Abarbeitung der TRAP-Routine gelesen wird.

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Interrupts sollten deshalb wie in einer der beiden nachfolgenden Alternativen behandelt werden.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGO go1, sig1int;
  WHILE TRUE DO
  ...
  ENDWHILE
ENDPROC
```

Sämtliche Aktivierungen von Interrupts erfolgen am Beginn des Programms. Diese Instruktionen werden dann außerhalb des Hauptablaufs des Programms gehalten.

```
VAR intnum sig1int;
PROC main ()
  CONNECT sig1int WITH iroutine1;
  ISignalGO go1, sig1int;
  ...
  IDelete sig1int;
ENDPROC
```

Der Interrupt wird am Ende des Programms gelöscht und dann erneut aktiviert. Beachten Sie in diesem Fall, dass der Interrupt für eine kurze Zeit inaktiv ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.119 ISignalGO - Anfordern von Interrupts von einer Gruppe digitaler Ausgangssignale

RobotWare - OS

Fortsetzung

Syntax

```
ISignalGO
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Signal ':=' ] < variable (VAR) of signalgo > ','
  [ Interrupt':=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt von einem analogen Ausgangssignal	*ISignalDO - Interrupts von einem digitalen Ausgangssignal auf Seite 324*
Interrupt von Gruppeneingangssignalen	*ISignalGI - Anfordern von Interrupts von einer Gruppe digitaler Eingangssignale auf Seite 328*
Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.120 ISleep - Deaktivieren eines Interrupts

Verwendung

`ISleep`(*Interrupt Sleep*) wird zum temporären Deaktivieren eines einzelnen Interrupts verwendet.

Für die Dauer der Deaktivierung wird jeder neu erzeugte Interrupt des angegebenen Typs ignoriert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ISleep`.

Siehe auch *Weitere Beispiele auf Seite 334*.

Beispiel 1:

```
ISleep sig1int;
```

Der Interrupt `sig1int` wird deaktiviert.

Argumente

```
ISleep Interrupt
```

`Interrupt`

Datentyp: `intnum`

Die Variable (Interrupt-Identität) des Interrupts.

Programmabarbeitung

Sämtliche generierte Interrupts des angegebenen Typs werden ignoriert, bis Interrupts erneut durch die Instruktion `IWatch` aktiviert werden. Alle Interrupts, die generiert werden, während `ISleep` wirksam ist, werden ignoriert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `ISleep` sind unten aufgeführt.

Beispiel 1:

```
VAR intnum timeint;
VAR iodev binfile;

PROC ISleep_example()
  CONNECT timeint WITH write_binfile;
  ITimer 6, timeint;
  !...
  ISleep timeint;
  WriteBin binfile, buffer, 30;
  WriteBin binfile, buffer2, 30;
  IWatch timeint;
  !...
ENDPROC

TRAP write_binfile
  WriteBin binfile, buffer3, 1;
ENDTRAP
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das Schreiben in die `binfile` erfolgt alle 6 Sekunden. Die Interrupts sind nicht zulässig, wenn die Kommunikation von der Main-Routine aktiv ist.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_UNKINO`	Die Interrupt-Nummer ist unbekannt. Interrupts, die weder angefordert noch freigegeben wurden, sind nicht zulässig.
`ERR_INOISSAFE`	Wenn versucht wird, einen sicheren Interrupt mit `ISleep` vorübergehend zu deaktivieren.

Syntax

```
ISleep
  [ Interrupt ':=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Freigeben eines Interrupts	*IWatch - Aktivieren eines Interrupts auf Seite 343*
Blockieren aller Interrupts	*IDisable - Blockieren von Interrupts auf Seite 253*
Abbrechen eines Interrupts	*IDelete - Abbrechen eines Interrupts auf Seite 252*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.121 ITimer - Anfordern eines zeitgesteuerten Interrupts

Verwendung

`ITimer` (*Interrupt Timer*) wird zum Anfordern und Freigeben eines zeitgesteuerten Interrupts verwendet.

Mithilfe dieser Instruktion kann z. B. der Status von Peripheriegeräten einmal pro Minute geprüft werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ITimer`:

Siehe auch *Weitere Beispiele auf Seite 337*.

Beispiel 1:

```
VAR intnum timeint;
PROC main()
  CONNECT timeint WITH iroutine1;
  ITimer 60, timeint;
```

Fordert einen Interrupt an, der zyklisch alle `60` Sekunden auftreten soll. Dann erfolgt ein Aufruf an die Interrupt-Routine `iroutine1`.

Beispiel 2:

```
ITimer \Single, 60, timeint;
```

Fordert einen Interrupt an, der einmal nach `60` Sekunden auftreten soll.

Argumente

```
ITimer [ \Single ] | [ \SingleSafe ] Time Interrupt
```

`[ \Single ]`

Datentyp: `switch`

Gibt an, ob der Interrupt nur einmal oder zyklisch auftreten soll.

Wenn das Argument `Single` gesetzt ist, tritt der Interrupt nur einmal auf. Wenn die Argumente `Single` und `SingleSafe` ausgelassen werden, tritt der Interrupt jedes Mal zur angegebenen Zeit auf.

`[ \SingleSafe ]`

Datentyp: `switch`

Gibt an, dass es sich um einen einzelnen sicheren Interrupt handelt. Die Definition von „einzeln“ finden Sie in der Beschreibung des `Single`-Arguments. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt.

`Time`

Datentyp: `num`

Die Zeit, die verstreichen muss, bis der Interrupt eintritt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Wert wird in Sekunden angegeben. Wenn `Single` oder `SingleSafe` gesetzt ist, darf die Dauer nicht unter 0,01 Sekunden liegen. Die entsprechende Zeit für zyklische Interrupts beträgt 0,1 Sekunden.

`Interrupt`

Datentyp: `intnum`

Die Variable (Interrupt-Identität) des Interrupts. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

Programmabarbeitung

Die entsprechende Interrupt-Routine wird gemäß der Interrupt-Reihenfolge automatisch zum vorgegebenen Zeitpunkt aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Wenn der Interrupt zyklisch auftritt, beginnt bei Auftreten des Interrupts eine neue Berechnung der Zeit.

Weitere Beispiele

Weitere Beispiele für die Instruktion `ITimer` sind unten aufgeführt.

Beispiel 1:

```
VAR intnum timeint;
VAR iodev binfile;

PROC ISleep_example()
  CONNECT timeint WITH write_binfile;
  ITimer 6, timeint;
  !...
  ISleep timeint;
  WriteBin binfile, buffer, 30;
  WriteBin binfile, buffer2, 30;
  IWatch timeint;
  !...
ENDPROC

TRAP write_binfile
  WriteBin binfile, buffer3, 1;
ENDTRAP
```

Das Schreiben in die `binfile` erfolgt alle 6 Sekunden. Die Interrupts sind nicht zulässig, wenn die Kommunikation von der Main-Routine aktiv ist.

Einschränkungen

Dieselbe Variable für Interrupt-Identität kann nicht mehrmals verwendet werden, ohne sie zuvor zu löschen. Siehe Instruktionen - `ISignalDI`.

Syntax

```
ITimer
  [ '\' Single ] | [ '\' SingleSafe ] ','
  [ Time ':=' ] < expression (IN) of num >','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.121 ITimer - Anfordern eines zeitgesteuerten Interrupts

RobotWare - OS

Fortsetzung

```
[ Interrupt' :=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.122 IVarValue - Anfordern eines Interrupts mit variablem Wert

Verwendung

`IVarValue` (*Interrupt Variable Value*) wird verwendet, um einen Interrupt anzuweisen und zu aktivieren, wenn sich der Wert einer Variablen ändert, auf die über die Sensorschnittstelle zugegriffen wird.

Mit dieser Instruktion können z. B. die Spaltbreiten von einem Nahtfolgesystem abgelesen werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IVarValue`:

Beispiel 1:

```
LOCAL PERS num
     adptVlt{25}:=[1,1.2,1.4,1.6,1.8,2,2.16667,2.33333,2.5,...];
LOCAL PERS num
     adptWfd{25}:=[2,2.2,2.4,2.6,2.8,3,3.16667,3.33333,3.5,...];
LOCAL PERS num
     adptSpd{25}:=10,12,14,16,18,20,21.6667,23.3333,25[,...];
LOCAL CONST num GAP_VARIABLE_NO:=11;
PERS num gap_value;
VAR intnum IntAdap;

PROC main()
  ! Setup the interrupt. The trap routine AdapTrp will be called
  ! when the gap variable with number ´GAP_VARIABLE_NO’ in the
  !sensor interface has been changed. The new value will be
  ! available in the PERS gp_value variable.
  ! Connect to the sensor device "sen1:" (defined in sio.cfg).
  SenDevice "sen1:";

  CONNECT IntAdap WITH AdapTrp;
  IVarValue "sen1:", GAP_VARIABLE_NO, gap_value, IntAdap;

  ! Start welding
  ArcL\On,*,v100,adaptSm,adaptWd,adaptWv,z10,tool\j\Track:=track;
  ArcL\On,*,v100,adaptSm,adaptWd,adaptWv,z10,tool\j\Track:=track;

ENDPROC

TRAP AdapTrap
  VAR num ArrInd;
  !Scale the raw gap value received
  ArrInd:=ArrIndx(gap_value);

  ! Update active welddata PERS variable ‘adaptWd’ with new data
  ! from the arrays of predefined parameter arrays. The scaled gap
  ! value is used as index in the voltage, wirefeed and
  ! speed arrays.
  adaptWd.weld_voltage:=adptVlt{ArrInd};
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.122 IVarValue - Anfordern eines Interrupts mit variablem Wert

Optical Tracking

Fortsetzung

```
    adaptWd.weld_wirefeed:=adptWfd{ArrInd};
    adaptWd.weld_speed:=adptSpd{ArrInd};

    !Request a refresh of AW parameters using the new data i adaptWd
    ArcRefresh;

ENDTRAP
```

Argumente

```
IVarValue device VarNo Value Interrupt [\Unit] [\DeadBand]
     [\ReportAtTool] [\SpeedAdapt] [\APTR]
```

`device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`VarNo`

Datentyp: `num`

Die Nummer der zu überwachenden Variablen.

`Value`

Datentyp: `num`

Eine `PERS` -Variable (Persistente), die den neuen Wert von `VarNo` enthält.

`Interrupt`

Datentyp: `intnum`

Die Variable (Interrupt-Identität) des Interrupts. Diese sollte zuvor über die Instruktion `CONNECT` mit einer Interrupt-Routine verbunden worden sein.

`[\Unit]`

Datentyp: `num`

Skalierungsfaktor, mit dem der Sensorwert für `VarNo` multipliziert wird, bevor er geprüft und in `Value` gespeichert wird.

`[\DeadBand]`

Datentyp: `num`

Liegt der vom Sensor zurückgegebene Wert von `Varno` innerhalb von `+/-DeadBand`, wird kein Interrupt generiert.

`[\ReportAtTool]`

Datentyp: `switch`

Das optionale Argument ist nur für vorauseilende Sensoren verfügbar, z. B. für Optical Tracking-Sensoren. Das Argument gibt an, dass der Wert der Variablen nicht gleichzeitig evaluiert werden soll, sondern wenn der Roboter-TCP seine Position erreicht, d. h. der Vorgriff ausgeglichen ist.

`[\SpeedAdapt]`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`\SpeedAdapt` ist ein Skalierungsfaktor, der zum Ändern der Prozessgeschwindigkeit in `Arc` und `Cap`-Instruktionen verwendet wird. Er wird mit dem Sensorwert für `VarNo` multipliziert gemäß:

Prozessgeschwindigkeit = `\SpeedAdapt` * `value(VarNo)`

`[\APTR]`

Datentyp: `switch`

Gibt an, dass das Abonnement dieser Variable mit dem At-Point-Tracker verbunden werden sollte, z. B. WeldGuide, angegeben im Argument `device`.

Programmabarbeitung

Die entsprechende Interrupt-Routine wird gemäß der Interrupt-Reihenfolge automatisch zum vorgegebenen Zeitpunkt aufgerufen. Nach ihrem Abschluss wird die Programmabarbeitung an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.

Einschränkungen

- Dieselbe Variable für Interrupt-Identität kann nicht öfter als 5 Mal verwendet werden, ohne sie zuvor zu löschen.
- Alle Interrupts, die mit `IVarValue` eingerichtet wurden, müssen nach dem Neustart der Steuerung neu eingestellt werden.

VORSICHT

Durch eine zu hohe Interrupt-Frequenz wird die gesamte RAPID-Abarbeitung blockiert.

Syntax

```
IVarValue
  [ device ':=' ] < expression (IN) of string> ','
  [ VarNo ':=' ] < expression (IN) of num > ','
  [ Value ':=' ] < persistent (PERS) of num > ','
  [ Interrupt ':=' ] < variable (VAR) of intnum > ','
  [ '\' Unit ':=' ] < expression (IN) of num > ','
  [ '\' DeadBand ':=' ] < expression (IN) of num > ','
  [ '\' ReportAtTool ] ','
  [ '\' SpeedAdapt ':=' ] < expression (IN) of num > ','
  [ '\' APTR ] ';'
```

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Zusammenfassung von Interrupts und Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Optical Tracking	*Application manual - Continuous Application Platform*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.122 IVarValue - Anfordern eines Interrupts mit variablem Wert

Optical Tracking

Fortsetzung

Information über	Siehe
Optical Tracking Arc	*Anwendungshandbuch - Arc und Arc Sensor*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.123 IWatch - Aktivieren eines Interrupts

Verwendung

`IWatch`(*Interrupt Watch*) wird verwendet, um einen Interrupt zu aktivieren, der zuvor angefordert, doch mit `ISleep` deaktiviert wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `IWatch`:

Siehe auch *Weitere Beispiele auf Seite 343*.

Beispiel 1:

```
IWatch sig1int;
```

Der Interrupt `sig1int`, der zuvor deaktiviert wurde, wird aktiviert.

Argumente

```
IWatch Interrupt
```

`Interrupt`

Datentyp: `intnum`

Die Variable (Interrupt-Identität) des Interrupts.

Programmabarbeitung

Aktiviert Interrupts des angegebenen Typs wieder. Alle Interrupts, die generiert wurden, während die Instruktion `ISleep` wirksam war, werden ignoriert.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_UNKINO`	Die Interrupt-Nummer ist unbekannt. Interrupts, die weder angefordert noch freigegeben wurden, sind nicht zulässig.

Weitere Beispiele

Weitere Beispiele für die Instruktion `IWatch` sind unten aufgeführt.

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalDI di1,1,sig1int;
  ...
  ISleep sig1int;
  weldpart1;
  IWatch sig1int;
```

Bei der Abarbeitung der Routine `weldpart1` sind keine Interrupts vom Signal `di1` zulässig.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.123 IWatch - Aktivieren eines Interrupts

RobotWare - OS

Fortsetzung

Syntax

```
IWatch
  [ Interrupt ':=' ] < variable (VAR) of intnum > ';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Deaktivieren eines Interrupts	*ISleep - Deaktivieren eines Interrupts auf Seite 334*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.124 Label - Zeilenname

Verwendung

`Label`wird für die Bezeichnung einer Programmzeile verwendet. Mithilfe der Instruktion `GOTO` kann dieser Name dann verwendet werden, um die Programmabarbeitung in derselben Routine zu versetzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Label`:

Beispiel 1:

```
GOTO next;
...
next:
```

Die Programmabarbeitung fährt mit der Instruktion fort, die auf `next` folgt.

Argumente

```
Label:
```

`Label`

Identifier

Der Name, den Sie der Zeile geben möchten.

Programmabarbeitung

Das Abarbeiten dieser Instruktion hat keinerlei Auswirkungen.

Einschränkungen

Das Label darf nicht gleich mit einem

- anderen Label oder einem beliebigen Datennamen
- in derselben Routine sein.

Ein Label verbirgt globale Daten und Routinen mit demselben Namen, die sich innerhalb dieser Routine befinden.

Syntax

```
<identifier>':'
```

Weitere Informationen

Information über	Siehe
Bezeichner	*Technisches Referenzhandbuch - RAPID Overview*
Versetzen der Programmabarbeitung an ein Label	*GOTO - Sprung zu einer anderen Instruktion auf Seite 240*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.125 Load - Laden eines Programmmoduls während der Abarbeitung

Verwendung

`Load` wird verwendet, um ein Programmmodul während der Abarbeitung in den Programmspeicher zu laden.

Das geladene Programmmodul wird den bereits bestehenden Modulen im Programmspeicher hinzugefügt.

Ein Programm- oder Systemmodul kann im statischen (Standard) oder dynamischen Modus geladen werden.

Sowohl statisch als auch dynamisch geladene Module können durch die Instruktion `UnLoad` entfernt werden.

Statischer Modus

Folgende Tabelle beschreibt, wie verschiedene Operationen statisch geladene Programm- oder Systemmodule beeinflussen.

Modultyp	PZ über FlexPendant auf „main“ setzen	Neues RAPID-Programm öffnen
Programmmodul	Kein Einfluss	Entfernen
Systemmodul	Kein Einfluss	Kein Einfluss

Dynamischer Modus

Folgende Tabelle beschreibt, wie verschiedene Operationen dynamisch geladene Programm- oder Systemmodule beeinflussen.

Modultyp	PZ über FlexPendant auf „main“ setzen	Neues RAPID-Programm öffnen
Programmmodul	Entfernen	Entfernen
Systemmodul	Entfernen	Entfernen

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Load`:

Siehe auch *Weitere Beispiele auf Seite 348*.

Beispiel 1:

```
Load \Dynamic, diskhome \File:="PART_A.MOD";
```

Lädt das Programmmodul `PART_A.MOD` aus `diskhome` in den Programmspeicher. `diskhome` ist die vordefinierte string-Konstante `"HOME:"`. Das Programmodul wird im dynamischen Modus geladen.

Beispiel 2:

```
Load \Dynamic, diskhome \File:="PART_A.MOD";
Load \Dynamic, diskhome \File:="PART_B.MOD" \CheckRef;
```

Lädt das Programmmodul `PART_A.MOD` in den Programmspeicher. Anschließend wird `PART_B.MOD` geladen. Wenn `PART_A.MOD` Verweise auf `PART_B.MOD` enthält, kann mit `\CheckRef` nur überprüft werden, ob nicht aufgelöste Verweise vorhanden sind, wenn das letzte Modul geladen wurde. Wenn `\CheckRef` für `PART_A.MOD` verwendet wird, tritt ein Verknüpfungsfehler auf, und das Modul wird nicht geladen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
Load [\Dynamic] FilePath [\File] [\CheckRef]
```

`[\Dynamic]`

Datentyp: `switch`

Der Schalter (switch) ermöglicht das dynamische Laden eines Moduls. Anderenfalls erfolgt das Laden im statischen Modus.

`FilePath`

Datentyp: `string`

Pfad und Name der Datei, die in den Programmspeicher geladen wird. Der Dateiname soll ausgeschlossen werden, wenn das Argument `\File` verwendet wird.

`[\File]`

Datentyp: `string`

Wenn der Dateiname im Argument `FilePath` ausgelassen wird, muss er mit diesem Argument definiert werden.

`[\CheckRef]`

Datentyp: `switch`

Überprüfung auf nicht aufgelöste Verweise in der Programmtask nach dem Laden des Moduls. Wenn der Parameter nicht verwendet wird, erfolgt keine Überprüfung auf nicht aufgelöste Verweise.

Programmabarbeitung

Die Programmabarbeitung wartet das Ende des Ladevorgangs ab, bevor sie mit der nächsten Instruktion fortfährt.

Nicht aufgelöste Verweise werden für den Ladevorgang immer akzeptiert, doch bei der Abarbeitung eines nicht aufgelösten Verweises tritt ein Laufzeitfehler auf, wenn Parameter `\CheckRef` nicht verwendet wird.

Nach dem Laden des Programmmoduls wird es eingebunden und initialisiert. Die Initialisierung des geladenen Moduls setzt alle Variablen auf Modulebene auf ihre Anfangswerte.

Wenn beim Ladevorgang ein Fehler, einschließlich nicht aufgelöster Verweise bei Verwendung des Schalters `\CheckRef`, auftritt, ist das geladene Modul nicht mehr im Programmspeicher verfügbar.

Für eine gute Programmstruktur, die sich leicht verstehen und pflegen lässt, sollten sämtliche Lade- und Entladevorgänge von Programmmodulen im Hauptmodul erfolgen, das stets während der Abarbeitung im Programmspeicher vorhanden ist.

Nachfolgendes Beispiel unter *Weitere Beispiele auf Seite 348* zeigt das Laden eines Programms, das eine Hauptprozedur zu einem Hauptprogramm (mit einer weiteren Hauptprozedur) enthält.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.125 Load - Laden eines Programmmoduls während der Abarbeitung

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_FILNOTFND	Die in der Instruktion Load angegebene Datei wurde nicht gefunden.
ERR_IOERROR	Es gibt ein Problem beim Lesen der Datei in der Instruktion Load.
ERR_PRGMEMFULL	Das Modul kann nicht geladen werden, weil der Programmspeicher voll ist.
ERR_LOADED	Das Modul ist bereits in den Programmspeicher geladen.
ERR_SYNTAX	Das geladene Modul enthält Syntaxfehler.
ERR_LINKREF	• Das geladene Modul verursacht schwerwiegende Verknüpfungsfehler. • Wenn Load mit dem Schalter \CheckRef verwendet wird, um eine Überprüfung auf Verweisfehler durchzuführen, und der Programmspeicher nicht aufgelöste Verweise enthält.

Beim Auftreten von einigen dieser Fehler wird das aktuelle Modul entladen und ist in der ERROR-Behandlungsroutine nicht verfügbar.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion Load sind unten aufgeführt.

Weitere allgemeine Beispiele

```
Load \Dynamic, "HOME:/DOORDIR/DOOR1.MOD";
```

Lädt das Programmmodul DOOR1.MOD aus HOME: aus dem Verzeichnis DOORDIR in den Programmspeicher. Das Programmmodul wird im dynamischen Modus geladen.

```
Load "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Wie oben, nur unterschiedliche Syntax. Das Modul wird im statischen Modus geladen.

```
Load\Dynamic, "HOME:/DOORDIR/DOOR1.MOD";
%"routine_x"%;
UnLoad "HOME:/DOORDIR/DOOR1.MOD";
```

Prozedur routine_x wird während der Abarbeitung gebunden (späte Bindung).

Geladenes Modul enthält Hauptprozedur

car.mod:

```
MODULE car
  PROC main()
    ...
    TEST part
    CASE door_part:
      Load \Dynamic, "HOME:/door.mod";
      %door:main%;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
      Unload "HOME:/door.mod";
    CASE window_part:
      Load \Dynamic, "HOME:/window.mod";
      %window:main%;
      Unload \Save "HOME:/window.mod";
    ENDTEST
  ENDPROC
ENDMODULE
```

door.mod:

```
MODULE door
  PROC main()
    ...
  ENDPROC
ENDMODULE
```

window.mod:

```
MODULE window
  PROC main()
    ...
  ENDPROC
ENDMODULE
```

Das Beispiel oben zeigt, wie Sie ein Modul laden können, das eine `main`-Prozedur enthält. Dieses Modul kann separat entwickelt und getestet und später unter Verwendung eines Hauptprogramms mit `Load` oder `StartLoad... WaitLoad` in das System geladen werden. In diesem Beispiel handelt es sich um `car.mod`, das andere Module wie `door.mod` oder `window.mod` lädt.

Im Modul `car.mod` laden Sie `door.mod` oder `window.mod` in `"HOME:"`. Weil die `main`-Prozeduren in `door.mod` und `window.mod` nach dem Laden vom System als **LOCAL** im Modul betrachtet werden, werden die Prozeduraufrufe auf folgende Weise ausgeführt: `%"door:main"%` oder `%"window: main"%`. Diese Syntax wird verwendet, wenn Sie auf **LOCAL**-Prozeduren in anderen Modulen zugreifen möchten, in diesem Beispiel auf `main` im Modul `door` oder im Modul `window`.

Durch das Entladen der Module mit dem Argument `\Save` werden die `main`-Prozeduren im gespeicherten Programm wieder global.

Wenn das Modul `car` oder `window` im System geladen ist und Sie den Programmzeiger aus einem beliebigen Teil des Programms auf `main` setzen, wird der Programmzeiger immer auf die globale Hauptprozedur `main` im Hauptmodul (`car.mod` in diesem Beispiel) gesetzt.

Einschränkungen

Verhindern Sie Roboterbewegungen während des Ladevorgangs.

Syntax

```
Load
  ['\'Dynamic',']
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.125 Load - Laden eines Programmmoduls während der Abarbeitung

RobotWare - OS

Fortsetzung

```
[FilePath':=']<expression (IN) of string>
['\'File':=' <expression (IN) of string>]
['\'CheckRef]';'
```

Weitere Informationen

Information über	Siehe
Entfernen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*
Laden eines Programmmoduls zeitgleich mit einer anderen Programmabarbeitung	*StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802* *WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*
Überprüft Programmreferenzen.	*CheckProgRef - Überprüfen von Programmverweisen auf Seite 131*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.126 LoadId - Lastidentifikation der Werkzeug- oder Nutzlast

Verwendung

`LoadId` (*Load Identification*) kann zur Lastidentifikation des Werkzeugs (auch des Greifwerkzeugs bei raumfestem TCP) oder der Nutzlast (wird über die Instruktion `GripLoad` aktiviert) verwendet werden, indem ein benutzerdefiniertes RAPID-Programm ausgeführt wird.

> **Hinweis**
>
> Eine einfachere Identifizierung der Werkzeug- oder Nutzlast ermöglicht das interaktive RAPID-Programm `LoadIdentify`. Dieses Programm kann über folgendes Menü gestartet werden: `ProgramEditor/Debug/Call Routine.../LoadIdentify`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `LoadId`:

Siehe auch *Weitere Beispiele auf Seite 355*.

Beispiel 1:

```
VAR bool invalid_pos := TRUE;
VAR jointtarget joints;
VAR bool valid_joints{12};
CONST speeddata low_ori_speed := [20, 5, 20, 5];
VAR bool slow_test_flag := TRUE;
PERS tooldata grip3 := [ TRUE, [[97.4, 0, 223.1], [0.924, 0, 0.383
     ,0]], [0, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0]];
! Check if valid robot type
IF ParIdRobValid(TOOL_LOAD_ID) <> ROB_LOAD_VAL THEN
  EXIT;
ENDIF
! Check if valid robot position
WHILE invalid_pos = TRUE DO
  joints := CJointT();
  IF ParIdPosValid (TOOL_LOAD_ID, joints, valid_joints) = TRUE THEN
    ! Valid position
    invalid_pos := FALSE;
  ELSE
    ! Invalid position
    ! Adjust the position by program movements (horizontal tilt
        house)
    MoveAbsJ joints, low_ori_speed, fine, tool0;
  ENDIF
ENDWHILE
! Do slow test for check of free working area
! Load modules into the system
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.126 LoadId - Lastidentifikation der Werkzeug- oder Nutzlast
RobotWare-OS
Fortsetzung

```
IF slow_test_flag = TRUE THEN
  %"LoadId"% TOOL_LOAD_ID, MASS_WITH_AX3, grip3 \SlowTest;
ENDIF
! Do measurement and update all load data in grip3
%"LoadID"% TOOL_LOAD_ID, MASS_WITH_AX3, grip3;
! Unload modules
UnLoad "RELEASE:/system/mockit.sys";
UnLoad "RELEASE:/system/mockit1.sys";
```

Lastidentifikation des Werkzeugs `grip3`.

Bedingung

Vor Lastmessungen mit `LoadId` sollten folgende Bedingungen erfüllt sein:

- Stellen Sie sicher, dass alle Lasten korrekt am Roboter befestigt sind.
- Prüfen Sie mit `ParIdRobValid`, ob es sich um einen gültigen Robotertyp handelt.
- Prüfen Sie mit `ParIdPosValid`, ob es sich um eine gültige Position handelt.
  - ob sich Achse 3, 5 und 6 nicht in der Nähe ihrer entsprechenden Arbeitsbereichsgrenzen befinden,
  - Kippgehäuse fast horizontal, d.h. dass Achse 4 in Nullposition ist
- Folgende Daten sollten in den Systemparametern und den Argumenten für `LoadId` definiert sein, bevor `LoadId` ausgeführt wird:

  Folgende Tabelle zeigt die Lastidentifikation des Werkzeugs.

Modi der Lastidentifikation / Daten definiert vor Ausführung von `LoadId`	Beweglicher TCP Masse bekannt	Beweglicher TCP Masse unbekannt	Raumfester TCP Masse bekannt	Raumfester TCP Masse unbekannt
Upper arm load (Systemparameter)		Definiert		Definiert
Masse an Werkzeug	Definiert		Definiert	

  Folgende Tabelle zeigt die Lastidentifikation der Nutzlast.

Modi der Lastidentifikation / Daten definiert vor Ausführung von `LoadId`	Beweglicher TCP Masse bekannt	Beweglicher TCP Masse unbekannt	Raumfester TCP Masse bekannt	Raumfester TCP Masse unbekannt
Upper arm load (Systemparameter)		Definiert		Definiert
Lastdaten an Werkzeug	Definiert	Definiert	Definiert	Definiert
Masse an Nutzlast	Definiert		Definiert	
Werkzeug-Koordinatensystem an Werkzeug	Definiert	Definiert		
Anwender-Koordinatensystem an Werkobjekt			Definiert	Definiert
Objekt-Koordinatensystem an Werkobjekt			Definiert	Definiert

- Betriebsmodus und Geschwindigkeits-Override:
  - Langsamer Test im Einrichtbetrieb mit reduzierter Geschwindigkeit

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Lastmessungen im Automatikbetrieb (oder im Einrichtbetrieb mit voller Geschwindigkeit) mit 100 % Geschwindigkeits-Override

Argumente

```
LoadId ParIdType LoadIdType Tool [\PayLoad] [\WObj] [\ConfAngle]
     [\SlowTest] [\Accuracy]
```

`ParIdType`

Datentyp: `paridnum`

Typ der Lastidentifikation wie in nachstehender Tabelle definiert.

Wert	Symbolische Konstante	Kommentar
1	TOOL_LOAD_ID	Identifizieren der Werkzeuglast
2	PAY_LOAD_ID	Identifizieren der Nutzlast (Ref.-Instruktion GripLoad)

`LoadIdType`

Datentyp: `loadidnum`

Typ der Lastidentifikation wie in nachstehender Tabelle definiert.

Wert	Symbolische Konstante	Kommentar
1	MASS_KNOWN	Bekannte Masse des Werkzeugs bzw. der Nutzlast (Masse des angegebenen Werkzeugs oder der angegebenen Nutzlast muss angegeben werden)
2	MASS_WITH_AX3	Unbekannte Masse des Werkzeugs bzw. der Nutzlast. Masse an Werkzeug oder Nutzlast wird bei Bewegung der Achse 3 festgestellt.

`Tool`

Datentyp: `tooldata`

Persistente Variable für das zu identifizierende Werkzeug. Wenn das Argument `\PayLoad` angegeben ist, die persistente Variable für das verwendete Werkzeug.

Für eine Lastidentifizierung des Werkzeugs sollten die Argumente `\PayLoad` und `\WObj` nicht angegeben werden.

`[ \ PayLoad ]`

Datentyp: `loaddata`

Persistente Variable für die zu identifizierende Nutzlast.

Dieses optionale Argument muss für die Lastidentifikation der Nutzlast immer angegeben werden.

`[ \ WObj ]`

Datentyp: `wobjdata`

Persistente Variable für das verwendete Werkobjekt.

Dieses optionale Argument muss für die Lastidentifikation der Nutzlast bei raumfestem TCP immer angegeben werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.126 LoadId - Lastidentifikation der Werkzeug- oder Nutzlast

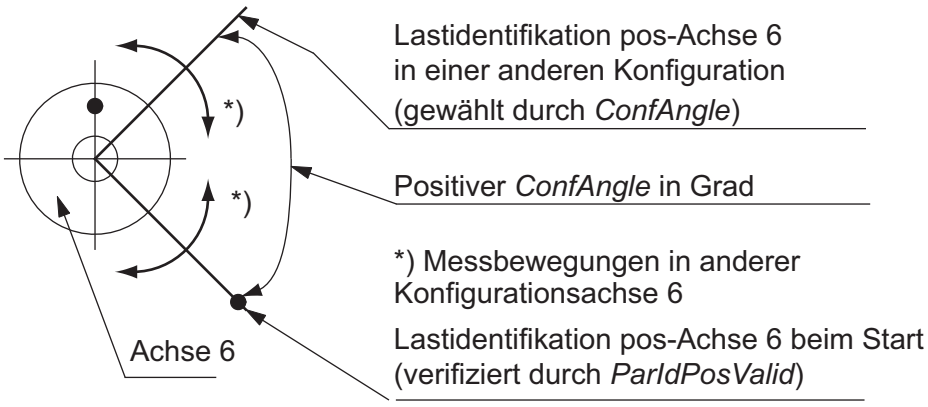
RobotWare-OS

Fortsetzung

`[ \ ConfAngle ]`

Datentyp: `num`

Optionales Argument für die Angabe eines bestimmten Konfigurationswinkels ± Grad, der zur Parameteridentifikation verwendet werden soll.



xx0500002198

Standard +90 Grad, wenn dieses Argument angegeben ist. Min. + oder - 30 Grad. Optimum + oder - 90 Grad.

`[ \ SlowTest ]`

Datentyp: `switch`

Optionales Argument zur Angabe, ob zur Überprüfung des freien Arbeitsbereichs nur ein langsamer Test durchgeführt werden soll. Siehe Tabelle unten:

`LoadId ... \SlowTest`	**Nur langsamen Test durchführen**
`LoadId ...`	**Nur Messung durchführen und Werkzeug oder Nutzlast aktualisieren**

`[ \ Accuracy ]`

Datentyp: `num`

Variable für die Ausgabe der berechneten Messgenauigkeit in % für die gesamte Lastidentifikationsberechnung (100 % bedeutet maximale Genauigkeit).

Programmabarbeitung

Der Roboter wird mit den Achsen 5 und 6 eine große Anzahl relativ kleiner Transport- und Messbewegungen ausführen. Um die Masse zu ermitteln, werden auch mit Achse 3 Bewegungen ausgeführt.

Nach allen Messungen, Bewegungen und Lastberechnungen werden die Lastdaten in das Argument `Tool` oder `PayLoad` ausgegeben. Folgende Lastdaten werden berechnet:

- **Masse in kg (falls die Masse unbekannt ist)**
- **Schwerpunkt (x, y, z) und Drehmomentachsen**
- **Trägheit ix, iy und iz in kgm**

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_PID_MOVESTOP	Bei jedem Fehler während der Ausführung der RAPID NOSTEPIN-Routine LoadId. Der Programmzeiger verweist auf den Benutzeraufruf von LoadId.
ERR_PID_RAISE_PP	
ERR_LOADID_FATAL	

Weitere Beispiele

Weitere Beispiele für die Instruktion LoadId sind unten aufgeführt.

Beispiel 1:

```
PERS tooldata grip3 := [ FALSE, [[97.4, 0, 223.1], [0.924, 0, 0.383
     ,0]], [6, [10, 10, 100], [0.5, 0.5, 0.5, 0.5], 1.2, 2.7,
     0.5]];
PERS loaddata piece5 := [ 5, [0, 0, 0], [1, 0, 0, 0], 0, 0, 0];
PERS wobjdata wobj2 := [ TRUE, TRUE, "", [ [34, 0, -45], [0.5,
     -0.5, 0.5 ,-0.5] ], [ [0.56, 10, 68], [0.5, 0.5, 0.5 ,0.5] ]
     ];
VAR num load_accuracy;
! Load modules into the system
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
! Do measurement and update all payload data except mass in piece5
%"LoadId"% PAY_LOAD_ID, MASS_KNOWN, grip3 \PayLoad:=piece5
     \WObj:=wobj2 \Accuracy:=load_accuracy;
TPWrite " Load accuracy for piece5 (%) = " \Num:=load_accuracy;
! Unload modules
UnLoad "RELEASE:/system/mockit.sys";
UnLoad "RELEASE:/system/mockit1.sys";
```

Lastidentifikation der Nutzlast piece5 mit bekannter Masse mit raumfestem TCP.

Einschränkungen

Normalerweise wird die Lastidentifikation eines Werkzeugs oder einer Nutzlast des Roboters über die Serviceroutine LoadIdentify durchgeführt. Diese Identifikation kann auch mit der RAPID-Instruktion LoadId durchgeführt werden. Vor dem Laden oder Abarbeiten des Programms mit LoadId müssen folgende Module in das System geladen werden:

```
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
```

Dann ist es möglich, LoadId mit einem Aufruf mit später Bindung aufzurufen (siehe obiges Beispiel 1).

Es ist nicht möglich, die Lastidentifikationsbewegungen nach einer beliebigen Art von Stopp wie beispielsweise einem Programmstopp, einem Nothalt oder einem Stromausfall wieder aufzunehmen. Sie müssen dann von Beginn an neu gestartet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.126 LoadId - Lastidentifikation der Werkzeug- oder Nutzlast

RobotWare-OS

Fortsetzung

Syntax

```
LoadId
  [ ParIdType ':=' ] <expression (IN) of paridnum> ',
  [ LoadIdType ':=' ] <expression (IN) of loadidnum> ','
  [ Tool ':=' ] <persistent (PERS) of tooldata>
  [ '\' PayLoad ':=' <persistent (PERS) of loaddata> ]
  [ '\' WObj ':=' <persistent (PERS) of wobjdata> ]
  [ '\' ConfAngle ':=' <expression (IN) of num> ]
  [ '\' SlowTest ]
  [ '\' Accuracy ':=' <variable (VAR) of num> ] ';'
```

Weitere Informationen

Information über	Siehe
Vordefiniertes Programm Load Identify	*Bedienungsanleitung - IRC5 mit FlexPendant*
Parameteridentifikationstyp	*paridnum - Parameteridentifikationstyp auf Seite 1749*
Ergebnis von `ParIdRobValid`	*paridvalidnum - Ergebnis von ParIdRobValid auf Seite 1751*
Lastidentifikationstyp	*loadidnum – Lastidentifikationstyp auf Seite 1729*
Gültiger Robotertyp	*ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation auf Seite 1424*
Gültige Roboterposition	*ParIdPosValid - Gültige Roboterposition für Parameteridentifikation auf Seite 1421*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.127 MakeDir - Erstellen eines neuen Verzeichnisses

Verwendung

`MakeDir` wird verwendet, um ein neues Verzeichnis zu erstellen. Der Benutzer muss für das übergeordnete Verzeichnis, unter dem das neue Verzeichnis erstellt werden soll, über eine Schreib- und Ausführungserlaubnis verfügen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MakeDir`:

Beispiel 1:

```
MakeDir "HOME:/newdir";
```

Dieses Beispiel erstellt ein neues Verzeichnis mit der Bezeichnung `newdir` unter `HOME`:

Argumente

```
MakeDir Path
```

`Path`

Datentyp:`string`

Der Name des neuen Verzeichnisses mit dem vollständigen oder relativen Pfad.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_FILEACC`	Das Verzeichnis kann nicht erstellt werden.

Syntax

```
MakeDir
[ Path':=' ] < expression (IN) of string>';'
```

Weitere Informationen

Information über	Siehe
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.127 MakeDir - Erstellen eines neuen Verzeichnisses

RobotWare - OS

Fortsetzung

Information über	Siehe
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.128 ManLoadIdProc - Lastidentifikation des Manipulators IRBP

Verwendung

`ManLoadIdProc` (*Manipulator Load Identification Procedure*) wird für die Lastidentifikation der Nutzlast für externe Manipulatoren verwendet, indem ein benutzerdefiniertes RAPID-Programm ausgeführt wird.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Hinweis

Eine einfachere Identifizierung der Nutzlast ermöglicht die Service-Routine **ManLoadIdentify**. Diese Service-Routine kann vom Menü **Program Editor**, **Debug**, **Call Routine**, **ManLoadIdentify** gestartet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ManLoadIdProc`:

```
PERS loaddata myload := [6,[0,0,0],[1,0,0,0],0,0,0];
VAR bool defined;
ActUnit STN1;
ManLoadIdProc \ParIdType := IRBP_L
  \MechUnit := STN1
  \PayLoad := myload
  \ConfigAngle := 60
  \AlreadyActive
  \DefinedFlag := defined;
DeactUnit STN1;
```

Lastidentifikation der Nutzlast `myload` an der mechanischen Einheit `STN1`. Der externe Manipulator ist vom Typ `IRBP-L`. Der Konfigurationswinkel wurde auf 60 Grad eingestellt. Der Manipulator wird vor der Lastidentifikation aktiviert und danach wieder deaktiviert. Nachdem die Identifizierung `myload` aktualisiert und definiert wurde, wird sie auf `TRUE` gesetzt.

Argumente

```
ManLoadIdProc [\ParIdType] [\MechUnit] | [\MechUnitName]
     [\AxisNumber] [\PayLoad] [\ConfigAngle] [\DeactAll] |
     [\AlreadyActive] [DefinedFlag] [DoExit]
```

`[ \ ParIdType ]`

Datentyp: `paridnum`

Parameteridentifikationstyp Vordefinierte Konstanten finden Sie unter dem Datentyp `paridnum`.

`[ \ MechUnit ]`

Datentyp: `mecunit`

Mechanische Einheit, die zur Lastidentifizierung verwendet wird. Kann nicht zusammen mit `\MechUnitName` verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.128 ManLoadIdProc - Lastidentifikation des Manipulators IRBP

RobotWare-OS

Fortsetzung

`[ \ MechUnitName ]`

Datentyp: `string`

Mechanische Einheit, die zur Lastidentifizierung, die als Zeichenfolge angegeben ist, verwendet wird. Kann nicht zusammen mit dem Argument `\MechUnit` **verwendet werden.**

`[ \ AxisNumber ]`

Datentyp: `num`

Die Achsennummer in der mechanischen Einheit, die die zu identifizierende Last hält.

`[ \ PayLoad ]`

Datentyp: `loaddata`

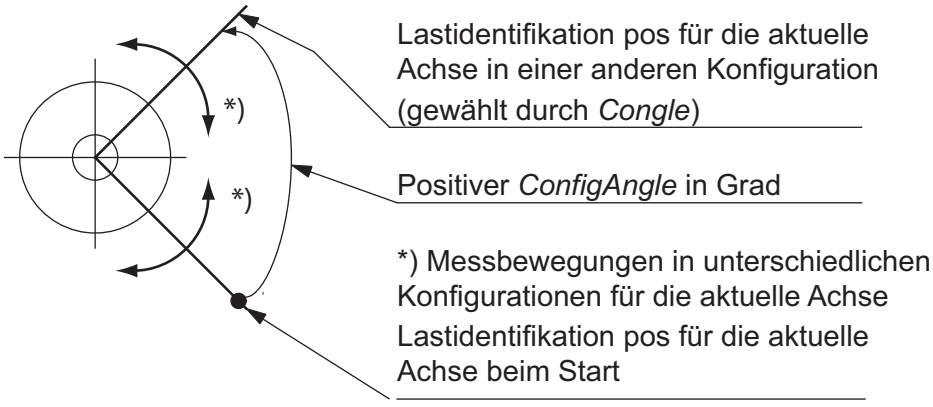
Variable für die zu identifizierende Nutzlast. Die Komponente "mass" muss angegeben werden.

Diese Variable wird im Anschluss an die Identifizierung aktualisiert.

`[ \ ConfigAngle ]`

Datentyp: `num`

Angabe eines bestimmten Konfigurationswinkels ± Grad, der zur Parameteridentifikation verwendet werden soll.



xx0500002197

Min. + oder - 30 Grad. Optimum + oder - 90 Grad.

`[ \ DeactAll ]`

Datentyp: `switch`

Falls dieser Datentyp verwendet wird, werden alle mechanischen Einheiten des Systems vor der Identifizierung deaktiviert. Anschließend wird die mechanische Einheit aktiviert, für die die Identifizierung durchgeführt werden soll. Er kann nicht zusammen mit dem Argument `\AlreadyActive` **verwendet werden.**

`[ \ AlreadyActive ]`

Datentyp: `switch`

Dieser Datentyp wird verwendet, wenn die mechanische Einheit, für die die Identifizierung durchgeführt werden soll, bereits aktiviert ist. Er kann nicht zusammen mit dem Argument `\DeactAll` **verwendet werden.**

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \ DefinedFlag ]`

Datentyp: `bool`

Dieses Argument wird auf `TRUE` gesetzt, wenn die Identifizierung durchgeführt wurde. Anderenfalls wird es auf `FALSE` gesetzt.

`[ \ DoExit]`

Datentyp: `bool`

Wenn `TRUE`, wird die Lastidentifizierung mit einem `EXIT`-Befehl beendet, um den Benutzer zu zwingen, den PZ auf „main“ zu setzen, bevor die Abarbeitung fortgesetzt wird. Wenn das Argument nicht vorhanden oder auf `FALSE` gesetzt ist, wird `EXIT` nicht ausgeführt. Mit `ManLoadIdProc` wird immer die aktuelle Bahn gelöscht.

Programmabarbeitung

Alle Argumente sind optional. Falls ein Argument nicht angegeben ist, wird der Benutzer vom FlexPendant nach dem Wert des Arguments gefragt (außer für `\DoExit`).

Folgende Werte müssen vom Benutzer stets angegeben werden: Masse und, für Manipulatoren des Typs IRBP R, z in mm.

Die mechanische Einheit wird eine große Anzahl relativ kurzer Transport- und Messbewegungen ausführen.

Nach allen Messungen, Bewegungen und Lastberechnungen werden die Lastdaten in das Argument `Payload` ausgegeben (falls es verwendet wird). Die folgenden Lastdaten werden berechnet.

Manipulatortyp/ Berechnete Lastdaten	IRBP-K	IRBP-L IRBP-C IRBP_T	IRBP-R	IRBP-A IRBP-B IRBP-D
Parameter `PayLoad - cog.x, cog.y, cog.z` in `loaddata` in mm	cog.x cog.y	cog.x cog.y	cog.x cog.y	cog.x cog.y cog.z
Parameter `PayLoad - ix, iy, iz` in `loaddata` in kgm^2	iz	iz	ix iy iz	ix iy iz

Die berechneten Daten werden am FlexPendant angezeigt.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_PID_MOVESTOP`	Bei jedem Fehler während der Ausführung der RAPID `NOSTEPIN`-Routine `ManLoadIdProc`. Der Programmzeiger verweist auf den Benutzeraufruf von `ManLoadIdProc`.
`ERR_PID_RAISE_PP`	
`ERR_LOADID_FATAL`	

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.128 ManLoadIdProc - Lastidentifikation des Manipulators IRBP

RobotWare-OS

Fortsetzung

Einschränkungen

Normalerweise wird die Lastidentifikation für externe Manipulatoren über die Serviceroutine `ManLoadIdentify` durchgeführt. Diese Identifikation kann auch mit der RAPID-Instruktion `ManLoadIdProc`.

Vor der Lastidentifizierung werden gegenwärtig ausgeführte Bahnen gelöscht. Der Programmzeiger geht nach der Lastidentifizierung verloren, wenn das Argument `\DoExit:=TRUE` verwendet wird.

Es ist nicht möglich, die Lastidentifikationsbewegungen nach einer beliebigen Art von Stopp wie beispielsweise einem Programmstopp, einem Nothalt oder einem Stromausfall wieder aufzunehmen. Sie müssen von Beginn an neu gestartet werden.

Syntax

```
ManLoadIdProc
  [ '\'ParIdType ':=' <expression (IN) of paridnum>]
  [ '\'MechUnit ':=' <variable (VAR) of mecunit> ]
  | ['\' MechUnitName ':=' <expression (IN) of string>]
  [ '\' AxisNumber ':=' <expression (IN) of num> ]
  [ '\' PayLoad ':=' <var or pers (INOUT) of loaddata>
  [ '\' ConfigAngle ':=' <expression (IN) of num>]
  [ '\' DeactAll] | [ '\' AlreadyActive]
  [ '\' DefinedFlag ':=' <variable (VAR) of bool> ]
  [ '\' DoExit ':=' <expression (IN) of bool> ] ';'
```

Weitere Informationen

Information über	Siehe
Parameteridentifikationstyp	*paridnum - Parameteridentifikationstyp auf Seite 1749*
Mechanische Einheit	*mecunit – Mechanische Einheit auf Seite 1731*
Nutzlast	*loaddata - Lastdaten auf Seite 1723*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.129 MatrixSolve - Lösen eines linearen Gleichungssystems

Verwendung

`MatrixSolve` wird zum Lösen linearer Gleichungssysteme in der Form *A*x=b* verwendet.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MatrixSolve`.

Siehe auch *Weitere Beispiele auf Seite 364*.

Beispiel 1:

```
VAR dnum A1{3,3}:=[[5, 2, 7],[-3, 1, 1],[1, 10, -3]];
VAR dnum b1{3}:=[-22, 39, 54];
VAR dnum x1{3};
...
MatrixSolve A1, b1, x1;
```

Das obige Beispiel löst das lineare Gleichungssystem. Das Datenfeld `x1` wird den Wert [-10, 7, 2] haben.

Beispiel 2:

```
VAR dnum A2{1,1} := [[5]];
VAR dnum b2{1}:=[35];
VAR dnum x2{1};
...
MatrixSolve A2, b2, x2;
```

Das obige Beispiel löst die triviale Gleichung *5x = 35*. Die Antwort ist 7.

Argumente

```
MatrixSolve A [\A_m] [\A_n] b x
```

`A`

Datentyp: `array of dnum`

`A` ist eine Matrix mit den Dimensionen *m * n*, wobei *m >= n*. Der Buchstabe `m` beschreibt die Anzahl der Zeilen, und der Buchstabe `n` beschreibt die Anzahl der Spalten der Matrix. Wenn *m > n* ist, wird das Gleichungssystem überbestimmt und eine Kleinste-Quadrate-Lösung wird zurückgegeben.

`[\A_m]`

Datentyp: `num`

Mit dem optionalen Argument `A_m` ist es möglich zu bestimmen, wie viele der Zeilen (`m`) in der Matrix `A` verwendet werden sollten.

`[\A_n]`

Datentyp: `num`

Mit dem optionalen Argument `A_n` ist es möglich zu bestimmen, wie viele der Spalten (`n`) in der Matrix `A` verwendet werden sollten.

`b`

Datentyp: `array of dnum`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.129 MatrixSolve - Lösen eines linearen Gleichungssystems
RobotWare - OS
Fortsetzung

b ist ein Feld mit der gleichen Dimension wie die Zeilen (m) der Matrix A. Bei Verwendung eines Datenfelds mit größeren Dimensionen als die Zeilen (m) der Matrix A werden die Komponenten oberhalb von m auf 0 gesetzt.

x

Datentyp: array of dnum

x ist ein Datenfeld mit der gleichen Dimension wie die Spalten (n) der Matrix A. Dies ist eine Datenfeld-Variable, in der das Ergebnis der Berechnung gespeichert wird. Bei Verwendung eines Datenfelds mit größeren Dimensionen als die Spalten (n) der Matrix A werden die Komponenten oberhalb von n auf 0 gesetzt.

Programmabarbeitung

MatrixSolve wird zum Lösen linearer Gleichungssysteme in der Form ***A*x=b*** verwendet. Wenn das System überbestimmt ist, wird eine Kleinste-Quadrate-Lösung zurückgegeben.

Wenn die optionalen Argumente A_m und A_n verwendet werden, ist es möglich, die gleiche Matrix für viele verschiedene Berechnungen zu verwenden, die unterschiedliche Größen der Matrix verwenden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_ARRAY_SIZE	Falsche Dimensionen oder falsche Werte für optionale Argumente werden verwendet.
ERR_MATRIX_SINGULAR	Die Eingabematrix ist singulär.

Weitere Beispiele

Weitere Beispiele für die Funktion MatrixSolve sind unten aufgeführt.

Beispiel 1:

```
VAR dnum A1{5,5}:=[[5,2,7,0,0], [-3,1,1,0,0], [1,10,-3,0,0]];
VAR dnum b1{8}:=[-22,39,54,0,0,0,0,0];
VAR dnum x1{8};

MatrixSolve A1 \A_m:=3 \A_n:=3, b1, x1;
```

Das obige Beispiel löst das lineare Gleichungssystem. Das Datenfeld x1 wird den Wert *[-10, 7, 2, 0, 0, 0, 0, 0]* haben. Dieses Beispiel ist das gleiche wie *Beispiel 1: auf Seite 363*. Der einzige Unterschied besteht darin, dass in diesem Beispiel dargestellt ist, wie die optionalen Argumente A_m und A_n verwendet werden, und dass größere Datenfelder als m und n für die Argumente b und x verwendet werden können. A_m und A_n können verwendet werden, um die Größe der Matrix zu begrenzen, so dass eine große allgemeine Matrix verwendet werden kann, um viele verschiedene Gleichungssysteme zu lösen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Bei der Lösung großer Matrizen kann es sein, dass der zugewiesene Speicher nicht ausreicht, um die aktuelle Berechnung abzuschließen, und es wird ein Ereignisprotokoll abgegeben. Die zugewiesene Speichergröße ist fest vorgegeben und kann nicht geändert werden. Versuchen Sie, kleinere Größen der Matrix zu lösen.

Syntax

```
MatrixSolve
  [ A ':=' ] < array {*}{*} expression (IN) of dnum > ','
  [ '\' A_m ':=' < expression (IN) of num > ]
  [ '\' A_n ':=' < expression (IN) of num > ] ','
  [ b ':=' ] < array {*} expression (IN) of dnum > ','
  [ x ':=' ] < array variable {*} (VAR) of dnum > ';'
```

Weitere Informationen

Information über	Siehe
Berechnen Sie eine QR-Faktorisierung.	*MatrixSolveQR - Berechnen einer QR-Faktorisierung auf Seite 366*
Berechnen Sie eine Einzelwertzerlegung.	*MatrixSVD - Berechnen einer Einzelwertzerlegung auf Seite 368*
Mathematische Instruktionen und Funktionen.	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.130 MatrixSolveQR - Berechnen einer QR-Faktorisierung

Verwendung

`MatrixSolveQR` wird verwendet, um eine QR-Faktorisierung einer (m x n) Matrix A zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MatrixSolveQR`.

Beispiel 1:

```
VAR dnum A4{3,3}:=[[12,-51,4], [6,167,-68], [-4,24,-41]];
VAR dnum Q4{3,3};
VAR dnum R4{3,3};
MatrixSolveQR A4, Q4, R4;
```

Instruktion `MatrixSolveQR` wird verwendet, um eine QR-Faktorisierung einer (m x n) Matrix A4 zu berechnen. Das Ergebnis der Berechnung ist:

```
Q4 := [[-0.857142857142857,0.394285714285714,0.331428571428571],
      [-0.428571428571429,-0.902857142857143,-0.0342857142857143],
      [0.285714285714286,-0.171428571428571,0.942857142857143]];

R4 := [[-14,-21,14], [0,-175,70], [0,0,-35]];
```

Argumente

```
MatrixSolveQR A [\A_m] [\A_n] Q R
```

A

Datentyp: `array of dnum`

A eine Matrix mit den Dimensionen m x n, wobei `m` die Anzahl der Zeilen ist und `n` die Anzahl der Spalten.

[\A_m]

Datentyp: `num`

Mit dem optionalen Argument `A_m` ist es möglich anzugeben, wie viele der Zeilen `m` in der Matrix `A` verwendet werden sollen.

[\A_n]

Datentyp: `num`

Mit dem optionalen Argument `A_n` ist es möglich anzugeben, wie viele der Spalten `n` in der Matrix `A` verwendet werden sollen.

Q

Datentyp: `array of dnum`

Orthogonale (m x m) Matrix. Dies ist eine Matrixvariable, mit der das Ergebnis der Berechnung gespeichert wird.

R

Datentyp: `array of dnum`

(m x n) der oberen Dreiecksmatrix. Dies ist eine Matrixvariable, mit der das Ergebnis der Berechnung gespeichert wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

`MatrixSolveQR` berechnet eine QR-Faktorisierung einer (m x n) Matrix A, so dass *A=Q*R*, wobei *Q* eine (m x m) orthogonale Matrix und *R* eine (m x n) obere Dreiecksmatrix ist.

Wenn die optionalen Argumente `A_m` und `A_n` verwendet werden, ist es möglich, die gleiche Matrix für viele verschiedene Berechnungen zu verwenden, die unterschiedliche Größen der Matrix verwenden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_ARRAY_SIZE`	Falsche Dimensionen oder falsche Werte für optionale Argumente werden verwendet.

Einschränkungen

Bei der Lösung großer Matrizen kann es sein, dass der zugewiesene Speicher nicht ausreicht, um die aktuelle Berechnung abzuschließen, und es wird ein Ereignisprotokoll abgegeben. Die zugewiesene Speichergröße ist fest vorgegeben und kann nicht geändert werden. Versuchen Sie, kleinere Größen der Matrix zu lösen.

Syntax

```
MatrixSolveQR
  [ A ':=' ] < array {*}{*} expression (IN) of dnum > ','
  [ '\' A_m ':=' < expression (IN) of num > ]
  [ '\' A_n ':=' < expression (IN) of num > ] ','
  [ Q ':=' ] < array variable {*}{*} (VAR) of dnum > ','
  [ R ':=' ] < array variable {*}{*} (VAR) of dnum > ';'
```

Weitere Informationen

Information über	Siehe
Lösen eines linearen Gleichungssystems.	*MatrixSolve - Lösen eines linearen Gleichungssystems auf Seite 363*
Berechnen Sie eine Einzelwertzerlegung.	*MatrixSVD - Berechnen einer Einzelwertzerlegung auf Seite 368*
Mathematische Instruktionen und Funktionen.	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.131 MatrixSVD - Berechnen einer Einzelwertzerlegung

Verwendung

`MatrixSVD` wird verwendet, um eine Einzelwertzerlegung (SVD) zu berechnen.

Die Einzelwertzerlegung (SVD) ist eine Faktorisierung einer realen Matrix, die in der Signalverarbeitung und Statistik viele nützliche Anwendungen hat.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MatrixSVD`.

Beispiel 1:

```
VAR dnum A3{7,5}:=[[32,5,30,-47,16], [41,46,-36,35,-33],
     [-38,47,-8,44,21], [42,-35,42,18,-47], [13,48,30,26,-23],
     [-41,46,46,25,-46], [-22,-1,16,-11,-41]];
VAR dnum U3{7,7};
VAR dnum S3{5};
VAR dnum V3{5,5};

MatrixSVD A3, U3, S3, V3;
```

Die Instruktion MatrixSVD wird zur Berechnung einer Einzelwertzerlegung verwendet. Das Ergebnis der Berechnung ist:

```
U3: = [ [-0.24489453114765, -0.241308890179438, -0.0602284681243788,
     -0.835993641906923, -0.0767894261551876, 0.240015157740137,
     0.340264541994411], [0.36884312087718, -0.011165754164993,
     0.807113814553714, -0.0174513269190971, -0.383355889416929,
     0.231289898525022, 0.107870591684966], [0.421298684810973,
     0.496825721418653, -0.0754892815524551, -0.0380766666594926,
     0.433352499740395, 0.0439085586237023, 0.615467956914505],
     [0.026022505267863, -0.748510046516493, 0.145750281133738,
     0.297910761636627, 0.491463032720705, 0.125969427524946,
     0.267688931284032], [0.44990665677733, -0.194363365765786,
     0.11415708386435, -0.406351234561054, 0.210650943912061,
     -0.684691075563818, -0.261683066861719], [0.634107267122289,
     -0.191796131356257, -0.458818389285907, -0.058536500937226,
     -0.126605540694026, 0.499569153226195, -0.285627519159172],
     [0.145957672500732, -0.24490688904148, -0.307330264960591,
     0.205089199291144, -0.597836567476733, -0.390807165865741,
     0.521598216836665]];

S3 := [128.223078192708, 106.345877681972, 86.7728210622664,
     62.5176992467654, 42.2777876032412];

V3 := [ [-0.241697068016687, -0.449209801318353, 0.774566517264602,
     -0.334347996967586, 0.16748495146732], [0.664865161158152,
     0.281358669789186, 0.148180746944642, -0.633450002424302,
     -0.235743880251818], [0.172346698575578, -0.571920245958167,
     -0.538598865014224, -0.371678506044732, 0.463648787880432],
     [0.541008847671309, 0.114564782377902, 0.291104463782398,
     0.449207178575608, 0.638479004558623], [-0.420883460142759,
     0.61549167032206, -0.0570844141501356, -0.383432536716582,
     0.541985217686462]];
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
MatrixSVD A [\A_m] [\A_n] U S V [\Econ]
```

A

Datentyp: `array of dnum`

A eine Matrix mit den Dimensionen m x n, wobei m die Anzahl der Zeilen und n die Anzahl der Spalten ist.

[\A_m]

Datentyp: `num`

Mit dem optionalen Argument A_m ist es möglich anzugeben, wie viele der Zeilen m in der Matrix A verwendet werden sollen.

[\A_n]

Datentyp: `num`

Mit dem optionalen Argument A_n ist es möglich anzugeben, wie viele der Spalten n in der Matrix A verwendet werden sollen.

U

Datentyp: `array of dnum`

U die linken Einzelvektoren von A, gespeichert als eine m x kk-Matrix, wobei kk zu den Spalten der Matrix A gleich ist (oder A_n), wenn der \Econ-Schalter verwendet wird, ansonsten ist der Wert gleich den Zeilen der Matrix A (oder A_m). Dies ist eine Matrixvariable, mit der das Ergebnis der Berechnung gespeichert wird. Bei Verwendung einer Matrix mit größeren Dimensionen als die Zeilen (m) und Spalten (n) der Matrix A werden die Komponenten über m und n auf 0 gesetzt.

S

Datentyp: `array of dnum`

S ein Feld mit der Dimension MIN(A m, A n) mit den Werten >= 0. Dies ist eine Datenfeldvariable, mit der das Ergebnis der Berechnung gespeichert wird. Wenn ein Datenfeld mit einer größeren Dimension als nötig verwendet wird, werden die Komponenten über n auf 0 gesetzt.

V

Datentyp: `array of dnum`

V die rechten Einzelvektoren von A, gespeichert als eine n x n-Matrix.

[\Econ]

(Economy size)

Datentyp: `switch`

Wenn \Econ und m > n verwendet werden, dann werden nur die n ersten Einzelvektoren von U berechnet.

Programmabarbeitung

MatrixSVD wird zur Berechnung einer Einzelwertzerlegung (SVD) der (m x n) Eingangsmatrix A verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.131 MatrixSVD - Berechnen einer Einzelwertzerlegung

RobotWare - OS

Fortsetzung

Ein SVD von einer Matrix A kann als A=U*S*V^T geschrieben werden, wobei U (m x m) (linke Einzelvektoren) ist, V (n x n) (rechte Einzelvektoren) ist und S eine (m x n) Diagonalmatrix mit nichtnegativen Elementen ist. Die Diagonalelemente sind die Einzelwerte von A.

Um Platz zu sparen, werden nur die Einzelwerte von S zurückgegeben und nicht die gesamte Matrix. Das heißt, S wird als ein Datenfeld mit der Länge MIN(m, n) dargestellt. Die Einzelwerte werden immer in absteigender Reihenfolge zurückgegeben.

Mit m > n kann weiterer Platz gespart werden, indem nur die n ersten singulären Vektoren von U berechnet werden. Dies wird durch den Schalter \Econ gesteuert. Daher gilt, wenn m > n und \Econ verwendet werden, dann ist U (m x n), andernfalls ist U (m x m).

Wenn die optionalen Argumente A_m und A_n verwendet werden, ist es möglich, die gleiche Matrix für viele verschiedene Berechnungen zu verwenden, die unterschiedliche Größen der Matrix verwenden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_ARRAY_SIZE	Falsche Dimensionen oder falsche Werte für optionale Argumente werden verwendet.

Einschränkungen

Bei der Lösung großer Matrizen kann es sein, dass der zugewiesene Speicher nicht ausreicht, um die aktuelle Berechnung abzuschließen, und es wird ein Ereignisprotokoll abgegeben. Die zugewiesene Speichergröße ist fest vorgegeben und kann nicht geändert werden. Versuchen Sie, kleinere Größen der Matrix zu lösen.

Syntax

```
MatrixSVD
  [ A ':=' ] < array {*}{*} expression (IN) of dnum > ','
  [ '\' A_m ':=' < expression (IN) of num > ]
  [ '\' A_n ':=' < expression (IN) of num > ] ','
  [ U ':=' ] < array variable {*}{*} (VAR) of dnum > ','
  [ S ':=' ] < array variable {*} (VAR) of dnum > ','
  [ V ':=' ] < array variable {*}{*} (VAR) of dnum > ','
  [ '\' Econ ] ';'
```

Weitere Informationen

Information über	Siehe
Lösen eines linearen Gleichungssystems.	*MatrixSolve - Lösen eines linearen Gleichungssystems auf Seite 363*
Berechnen einer QR-Faktorisierung	*MatrixSolveQR - Berechnen einer QR-Faktorisierung auf Seite 366*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Mathematische Instruktionen und Funktionen.	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.132 MechUnitLoad - Definieren einer Nutzlast für eine mechanische Einheit

Verwendung

`MechUnitLoad` wird verwendet, um eine Nutzlast für eine externe mechanische Einheit, beispielsweise für einen Positionierer, zu definieren. Die Nutzlast für einen Roboter wird mit der Instruktion `GripLoad` definiert.

Um eine optimale Bewegung zu erreichen, sollte diese Instruktion für alle mechanischen Einheiten mit dem dynamischen Modell (ABB-Positionierer und Bahnanwendungen) verwendet werden.

Die Instruktion `MechUnitLoad` sollte stets nach der Instruktion `ActUnit` abgearbeitet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Beschreibung

`MechUnitLoad` spezifiziert, welche Lasten die mechanischen Einheiten tragen. Spezifizierte Lasten werden im Steuerungssystem verwendet, damit die Bewegungen der mechanischen Einheiten ideal gesteuert werden können.

Die Nutzlast wird mit der Instruktion `MechUnitLoad` aufgenommen/abgelegt, die das Gewicht der Nutzlast dem Gewicht der mechanischen Einheit hinzufügt oder von ihm abzieht.

WARNUNG

Die korrekte aktuelle Werkzeuglast sowie, falls vorhanden, die Nutzlast des Roboters müssen unbedingt definiert werden. Falsche Definitionen der Lastdaten können zu Überlastung der mechanischen Roboterstruktur führen. Es besteht zudem ein Risiko, dass die Geschwindigkeit im manuell reduzierten Geschwindigkeitsmodus überschritten werden kann.

Die Angabe falscher Lastdaten kann folgende Konsequenzen haben:

- Der Roboter verwendet möglicherweise nicht seine maximale Kapazität.
- Die Bahngenauigkeit wird beeinträchtigt und es besteht das Risiko des Überschwingens.
- Die mechanische Struktur kann überlastet werden.

Die Steuerung überwacht ununterbrochen die Last und schreibt ein Ereignisprotokoll, wenn die Last höher als erwartet ist. Dieses Ereignisprotokoll wird gespeichert und Steuerungsspeicher protokolliert.

WARNUNG

Die obige Warnung gilt auch, wenn Nutzlasten für eine externe mechanische Einheit definiert werden.

Fortsetzung auf nächster Seite

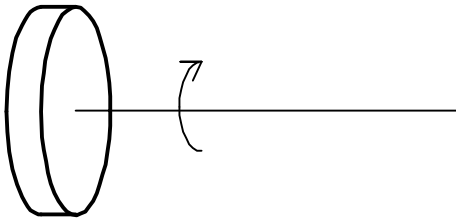
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MechUnitLoad`:

Abbildung

Nachstehende Abbildung zeigt Achse 1 an einer mechanischen Einheit namens `STN1` vom Typ IRBP L.



xx0500002142

Beispiel 1:

```
ActUnit SNT1;
MechUnitLoad STN1, 1, load0;
```

Aktiviert die mechanische Einheit `STN1` und definiert die Nutzlast `load0`, die keiner auf Achse 1 montierten Last entspricht.

Beispiel 2:

```
ActUnit STN1;
MechUnitLoad STN1, 1, fixture1;
```

Aktiviert die mechanische Einheit `STN1` und definiert die Nutzlast `fixture1`, die der auf Achse 1 montierten Befestigung entspricht.

Beispiel 3

```
ActUnit STN1;
MechUnitLoad STN1, 1, workpiece1;
```

Aktiviert die mechanische Einheit `STN1` und definiert die Nutzlast `workpiece1` die der auf Achse 1 montierten Befestigung/dem Werkstück entspricht.

Argumente

```
MechUnitLoad MechUnit AxisNo Load
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`AxisNo`

Axis Number

Datentyp: `num`

Die Achsennummer in der mechanischen Einheit, die die Last hält. Die Achsennummerierung beginnt mit 1.

`Load`

Datentyp: `loaddata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.132 MechUnitLoad - Definieren einer Nutzlast für eine mechanische Einheit

RobotWare - OS

Fortsetzung

Die Lastdaten, die die zu definierende aktuelle Nutzlast beschreiben, d. h. die Befestigung oder die Befestigung zusammen mit dem Werkstück, hängen davon ab, ob das Werkstück an der mechanischen Einheit montiert ist oder nicht.

Programmabarbeitung

Die angegebene Last gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue `MechUnitLoad`-Instruktion ausgeführt wird.

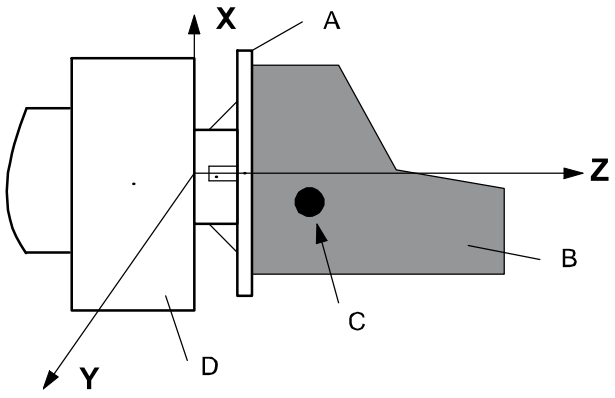
Nach der Abarbeitung von `MechUnitLoad`, wenn der Roboter und die zusätzlichen Achsen stillstehen, wird die spezifische Last für die angegebene mechanische Einheit und die angegebene Achse definiert. Dies bedeutet, dass die Nutzlast von der Steuerung kontrolliert und überwacht wird.

Die Standardnutzlast beim Neustartmodus **Reset System**(System zurücksetzen) ist für einen bestimmten mechanischen Einheitentyp die vordefinierte maximale Nutzlast.

Wenn eine andere Nutzlast verwendet wird, sollte die aktuelle Nutzlast für die mechanische Einheit und Achse in dieser Instruktion neu definiert werden. Das sollte immer nach Aktivierung der mechanischen Einheit erfolgen.

Die definierte Nutzlast bleibt nach einem Neustart erhalten. Die definierte Nutzlast bleibt bei einem Neustart des Programms nach einer manuellen Aktivierung anderer mechanischer Einheiten über das Bewegen-Fenster erhalten.

Die folgende Grafik zeigt eine Nutzlast am Endeffektor einer mechanischen Einheit (Endeffektor-Koordinatensystem für die mechanische Einheit).



xx0500002143

A	Endeffektor
B	Befestigung und Werkstück
C	Schwerpunkt für die Nutzlast (Halterung + Werkstück)
D	Mechanische Einheit

Fortsetzung auf nächster Seite

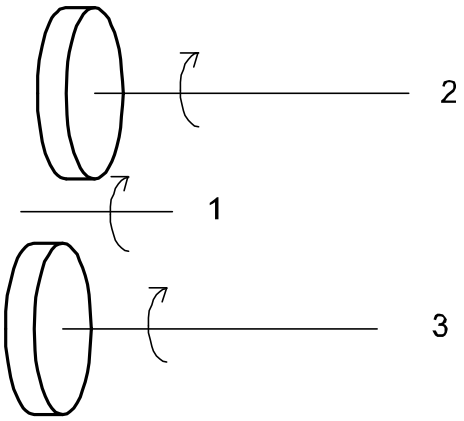
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `MechUnitLoad` sind unten aufgeführt.

Abbildung

Folgende Abbildung zeigt die mechanische Einheit `INTERCH` vom Typ IRBP K mit drei Achsen (1,2 und 3).



xx0500002144

Beispiel 1:

```
MoveL homeside1, v1000, fine, gun1;
...
ActUnit INTERCH;
```

Die vollständige mechanische Einheit `INTERCH` wird aktiviert.

Beispiel 2:

```
MechUnitLoad INTERCH, 2, workpiece1;
```

Definiert die Nutzlast `workpiece1` an der mechanischen Einheit `INTERCH` Achse 2.

Beispiel 3

```
MechUnitLoad INTERCH, 3, workpiece2;
```

Definiert die Nutzlast `workpiece2` an der mechanischen Einheit `INTERCH` Achse 3.

Beispiel 4:

```
MoveL homeside2, v1000, fine, gun1;
```

Die Achsen der mechanischen Einheit `INTERCH` bewegen sich an die Schalterposition `homeside2` mit montierter Nutzlast auf Achse 2 und 3.

Beispiel 5:

```
ActUnit STN1;
MechUnitLoad STN1, 1, workpiece1;
```

Die mechanische Einheit `STN1` wird aktiviert. Definiert die Nutzlast `workpiece1` an der mechanischen Einheit `STN1` Achse 1.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.132 MechUnitLoad - Definieren einer Nutzlast für eine mechanische Einheit

RobotWare - OS

Fortsetzung

Einschränkungen

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (`zonedata fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`MechUnitLoad` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

Syntax

```
MechUnitLoad
  [MechUnit ':=' ] <variable (VAR) of mecunit> ','
  [AxisNo ':=' ] <expression (IN) of num> ','
  [Load ':=' ] <persistent (PERS) of loaddata> ';'
```

Weitere Informationen

Information über	Siehe
Identifizierung einer Nutzlast für externe mechanische Einheiten	*Produkthandbuch - IRBP /D2009*
Definition der Daten einer mechanischen Einheit	*mecunit – Mechanische Einheit auf Seite 1731*
Definition der Lastdaten	*loaddata - Lastdaten auf Seite 1723*
Definieren der Nutzlast für den Roboter	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.133 MotionProcessModeSet - Einstellen des Bewegungsprozess-Modus

Verwendung

`MotionProcessModeSet` wird verwendet um den Bewegungsprozess-Modus (*Motion Process Mode*) für einen TCP-Roboter einzustellen.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MotionProcessModeSet`:

```
MotionProcessModeSet OPTIMAL_CYCLE_TIME_MODE;
! Do cycle-time critical movement
..
MotionProcessModeSet ACCURACY_MODE;
! Do cutting with high accuracy
..
```

Ändern des Bewegungsprozess-Modus des TCP-Roboters in der Laufzeit.

Argumente

```
MotionProcessModeSet Mode
```

`Mode`

Datentyp: `motionprocessmode`

Dafür den Bewegungsprozess-Modus verwenden. Es ist eine ganzzahlige Konstante mit dem Datentyp `motionprocessmode`.

Programmabarbeitung

Der Bewegungsprozess-Modus gilt für den TCP-Roboter, bis eine neue `MotionProcessModeSet` Instruktion ausgeführt wird, siehe *Weitere Informationen auf Seite 378*.

Der Standardwert für Bewegungsprozess-Modi wird automatisch eingestellt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.133 MotionProcessModeSet - Einstellen des Bewegungsprozess-Modus

Advanced Robot Motion

Fortsetzung

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `motionprocessmode` sind vordefiniert und können verwendet werden, um die Angabe der ganzen Zahl im Argument `Mode` zu spezifizieren.

Der Standardmodus ist durch den Systemparameter *Use Motion Process Mode* im Typ *Robot* und der Parametergruppe *Motion* definiert.

Symbolische Konstante	Konstanter Wert	Beschreibung
`OPTIMAL_CYCLE_TIME_MODE`	1	Dieser Modus sorgt für die kurzmöglichste Zykluszeit.
`LOW_SPEED_ACCURACY_MODE`	2	Dieser Modus verbessert die Bahngenauigkeit vor allem für große Roboter.
`LOW_SPEED_STIFF_MODE`	3	Dieser Modus wird für eine langsame Kontakt- und Anpassungsgeschwindigkeit empfohlen, bei der eine maximale Servo-Steifigkeit relevant ist.
`ACCURACY_MODE`	4	Dieser Modus verbessert die Bahngenauigkeit vor allem für kleine Roboter.
`MPM_USER_MODE_1`	5	Benutzerdefinierte Modi.
`MPM_USER_MODE_2`	6	
`MPM_USER_MODE_3`	7	
`MPM_USER_MODE_4`	8	
`PRESS_TENDING_MODE`	9	Vorrangig für flexible Greifer bei der Pressenbedienungsanwendung bestimmt.

Einschränkungen

Der Modus kann nur geändert werden, wenn der Roboter still steht, andernfalls wird ein Feinpunkt erzwungen.

Syntax

```
MotionProcessModeSet
  [Mode ':='] <expression (IN) of motionprocessmode>';'
```

Weitere Informationen

Information über	Siehe
Advanced robot motion	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfiguration der Parameter *Motion Process Mode*	*Technisches Referenzhandbuch - Systemparameter*
Abstimmen von Servos.	*TuneServo - Abstimmen von Servos auf Seite 1004*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.134 MotionSup - Deaktivieren/Aktivieren der Bewegungsüberwachung

Verwendung

`MotionSup` (*Motion Supervision*) wird verwendet, um die Bewegungsüberwachung für Roboterbewegungen bei der Programmabarbeitung zu deaktivieren oder zu aktivieren.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Beschreibung

Bewegungsüberwachung ist die Bezeichnung für eine Reihe von Funktionen für eine hochempfindliche, modellbasierte Überwachung des Roboters. Sie enthält die Funktion für die Lade- und Kollisionsüberwachung der Achsen und für die Kollisionserkennung. Da die Überwachungsfunktion sehr empfindlich ausgelegt ist, kann sie unter Umständen ausgelöst werden, wenn große Prozesskräfte auf den Roboter wirken.

Wenn die Last nicht richtig definiert ist, verwenden Sie die Serviceroutine zur Lasterkennung, um sie anzugeben. Wenn bei Anwendungen über einen längeren Zeitraum große externe Prozesskräfte wirken (z. B. beim Entgraten), können Sie die Überwachungsstufe der Bewegungsüberwachung mithilfe der Systemparameter anheben, bis sie nicht mehr ausgelöst wird. Wenn die externen Kräfte jedoch nur zeitweise wirken, wie beispielsweise beim Schließen einer großen Punktschweißzange, sollten Sie die Instruktion `MotionSup` verwenden, um für diejenigen Stellen der Anwendung, an denen die Störung auftritt, die Überwachungsstufe anzuheben (oder die Funktion auszuschalten).

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MotionSup`:

Beispiel 1:

```
! If the motion supervision is active in the system parameters,
! then it is active by default during program execution
...
! If motion supervision is deactivated in the system parameters
! then it cannot be activated using the MotionSup instruction
...
! Deactivate motion supervision during program execution
MotionSup \Off;
...
! Activate motion supervision again during program execution
MotionSup \On;
...
! Tune the supervision level to 200% (makes the function less
! sensitive) of the level in
! the system parameters
MotionSup \On \TuneValue:= 200;
...
! Activate motion supervision again.
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.134 MotionSup - Deaktivieren/Aktivieren der Bewegungsüberwachung

Collision Detection

Fortsetzung

```
! No back off at a motion collision
MotionSup \On \NoBackoff;
```

Argumente

```
MotionSup[\On] | [\Off] [\TuneValue] [\NoBackoff]
```

[\On]

Datentyp: switch

Aktivieren der Bewegungsüberwachungsfunktion bei der Programmabarbeitung (falls sie über die Systemparameter bereits aktiviert wurde).

[\Off]

Datentyp: switch

Deaktivieren der Bewegungsüberwachungsfunktion bei der Programmabarbeitung.

Eines der Argumente \On oder \Off muss angegeben werden.

[\TuneValue]

Datentyp: num

Einstellen der Empfindlichkeitsstufe der Bewegungsüberwachung in Prozent (1 bis 300 %) der Systemparameterstufe. Eine höhere Stufe führt zu einer geringeren Empfindlichkeit. Dieses Argument kann nur mit dem Argument \On kombiniert werden.

[\NoBackoff]

Datentyp: switch

Wenn dieser Schalter verwendet wird, bewegt sich der Roboter bei einer Bewegungskollision nicht rückwärts. Dieses Argument kann nur mit dem Argument \On kombiniert werden.

Programmabarbeitung

Die angegebene Bewegungsüberwachung gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue MotionSup-Instruktion ausgeführt wird.

Wenn die Bewegungsüberwachungsfunktion sowohl in den Systemparametern als auch im RAPID-Programm aktiviert ist und aufgrund einer Kollision ausgelöst wird, geschieht Folgendes:

- Der Roboter hält so schnell wie möglich an.
- der Roboter bewegt sich zurück, um vorhandene Restenergien abzubauen (wenn der Schalter \NoBackoff bei der letzten MotionSup-Instruktion nicht verwendet wurde)
- Die Programmabarbeitung stoppt mit einer Fehlermeldung.

Wenn die Bewegungsüberwachung in den Systemparametern aktiviert ist, ist sie bei der Programmabarbeitung standardmäßig aktiviert (TuneValue:=100 und bewegt sich zurück, um vorhandene Restenergien abzubauen). Diese Werte werden automatisch eingestellt.

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

- Die Bewegungsüberwachung ist nie aktiviert für externe Achsen oder beim Betrieb einer oder mehrerer Achsen im unabhängigen Achsenmodus. Wenn der Roboter im Softservo-Modus betrieben wird, kann es erforderlich sein, die Bewegungsüberwachung auszuschalten, um ein versehentliches Auslösen zu vermeiden.
- Wenn die Bewegungsüberwachung in den Systemparametern deaktiviert ist, kann sie nicht mit der MotionSup-Instruktion aktiviert werden.

Syntax

```
MotionSup
[ '\' On] | [ '\' Off ]
[ '\' Tunevalue':='< expression (IN) of num> ]
[ '\' NoBackoff ] ';'
```

Weitere Informationen

Information über	Siehe
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Allgemeine Beschreibung der Funktion	*Technisches Referenzhandbuch - RAPID Overview*
Einstellen mithilfe der Systemparameter	*Technisches Referenzhandbuch - Systemparameter*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.135 MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition

Verwendung

`MoveAbsJ` (*Move Absolute Joint*) wird verwendet, um den Roboter an eine absolute Position zu bewegen, die in den Achsenpositionen definiert ist.

Beispiele:

- Endpunkt ist ein singulärer Punkt
- für mehrdeutige Positionen am Modell IRB 6400C, z. B. für Bewegungen des Werkzeugs über dem Roboter.

Die letztendliche Position des Roboters während einer Bewegung mit `MoveAbsJ` wird weder durch das verwendete Werkzeug und Werkobjekt noch durch die aktive Programmverschiebung beeinflusst. Der Roboter verwendet diese Daten zur Berechnung der Last, der TCP-Geschwindigkeit und der Zonenbahn. Dieselben Werkzeuge können in mehreren Bewegungsanweisungen verwendet werden.

Der Roboter und die externen Achsen bewegen sich entlang einer nicht linearen Bahn an die Zielposition. Alle Achsen erreichen die Zielposition gleichzeitig.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveAbsJ`:

Siehe auch *Weitere Beispiele auf Seite 386*.

Beispiel 1:

```
MoveAbsJ p50, v1000, z50, tool2;
```

Der Roboter mit dem Werkzeug `tool2` wird entlang einer nicht linearen Bahn an die absolute Achsenposition `p50` mit der Geschwindigkeit `v1000` und den Zonendaten `z50` bewegt.

Beispiel 2:

```
MoveAbsJ *, v1000\T:=5, fine, grip3;
```

Der Roboter mit dem Werkzeug `grip3` wird entlang einer nicht linearen Bahn an einen Stopppunkt bewegt, der als absolute Achsenposition in der Instruktion gespeichert ist (gekennzeichnet durch *). Die vollständige Bewegung dauert `5` Sekunden.

Argumente

```
MoveAbsJ [\Conc] ToJointPos [\ID] [\NoEOffs] Speed [\V] | [\T] Zone
    [\Z] [\Inpos] Tool [\WObj] [\TLoad]
```

`[\Conc]`

Concurrent

Datentyp:`switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Das Argument wird normalerweise nicht verwendet. Es kann jedoch verwendet werden, um die Zyklusdauer zu verkürzen, z. B. bei der Kommunikation mit externen Geräten, wenn keine Synchronisierung erforderlich ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Mit dem Argument \Conc ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der StorePath-RestoPath enthält, sind Bewegungsinstruktionen mit dem Argument \Conc nicht zulässig.

Wenn dieses Argument ausgelassen wird und der ToJointPos kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

ToJointPos

To Joint Position

Datentyp: jointtarget

Der Zielpunkt als absolute Achsenposition des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

[\ID]

Synchronization id

Datentyp: identno

Das Argument [\ID] ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

[\NoEOffs]

No External Offsets

Datentyp: switch

Wenn das Argument \NoEOffs gesetzt ist, wird die Bewegung mit MoveAbsJ nicht durch aktive Offsets für externe Achsen beeinflusst.

Speed

Datentyp: speeddata

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

[\V]

Velocity

Datentyp: num

Dieses Argument wird verwendet, um die Geschwindigkeit des TCP in mm/s direkt in der Instruktion zu definieren. Es ersetzt dann die entsprechende Geschwindigkeit, die in den Geschwindigkeitsdaten angegeben ist.

[\T]

Time

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.135 MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition
RobotWare - OS
Fortsetzung

Datentyp: num

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Zone

Datentyp:zonedata

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

[\Z]

Zone

Datentyp: num

Dieses Argument wird verwendet, um die Position des Roboter-TCP direkt in der Instruktion anzugeben. Die Länge der Zonenbahn wird in mm angegeben und ersetzt dann die entsprechende Zone in den Zonendaten.

[\Inpos]

In position

Datentyp: stoppointdata

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP im Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter Zone angegeben ist.

Tool

Datentyp: tooldata

Das verwendete Werkzeug während der Bewegung.

Die Position des TCP und der Last auf dem Werkzeug werden in den Werkzeugdaten definiert. Die TCP-Position wird verwendet, um die Geschwindigkeit und die Zonenbahn für die Bewegung zu berechnen.

[\WObj]

Work Object

Datentyp: wobjdata

Das verwendete Werkzeug während der Bewegung.

Dieses Argument kann ausgelassen werden, wenn das Werkzeug vom Roboter gehalten wird. Wenn der Roboter das Werkobjekt hält, d. h. wenn das Werkzeug stationär ist, oder wenn koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

Im Fall des stationären Werkzeugs oder der koordinierten externen Achsen berechnet das System anhand der für das Werkobjekt definierten Daten die Geschwindigkeit und die Zonenbahn für die Bewegung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Eine Bewegung mit `MoveAbsJ` wird nicht durch aktive Programmverschiebung beeinflusst, und wenn sie mit dem Schalter `\NoEOffs` ausgeführt wird, erfolgt kein Offset für externe Achsen. Ohne den Schalter `\NoEOffs` werden die externen Achsen am Ziel durch aktive Offsets für externe Achsen beeinflusst.

Der Werkzeugarbeitspunkt wird mit Interpolation der Achsenwinkel an den absoluten Achsenzielpunkt bewegt. Dies bedeutet, dass jede Achse mit einer konstanten Achsengeschwindigkeit bewegt wird und dass alle Achsen gleichzeitig die Achsenzielposition erreichen, wodurch sich eine nicht lineare Bahn ergibt.

Allgemein gesagt bewegt sich der TCP in der ungefähren programmierten Geschwindigkeit. Das Werkzeug wird umorientiert und die externen Achsen werden gleichzeitig mit dem TCP bewegt. Wenn es nicht möglich ist, die programmierte Geschwindigkeit für die Umorientierung oder für die externen Achsen beizubehalten, wird die Geschwindigkeit des TCP reduziert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.135 MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition
RobotWare - OS
Fortsetzung

Eine Zonenbahn wird gewöhnlich generiert, wenn Bewegung an den nächsten Bahnabschnitt übertragen wird. Falls in den Zonendaten ein Stopppunkt angegeben ist, fährt die Programmabarbeitung nur fort, wenn der Roboter und die externen Achsen die korrekte Achsenposition erreicht haben.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CONC_MAX	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument \Conc verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion MoveAbsJ sind unten aufgeführt.

Beispiel 1:

```
MoveAbsJ *, v2000\V:=2200, z40 \Z:=45, grip3;
```

Das Werkzeug grip3 wird entlang einer nicht linearen Bahn an eine absolute Achsenposition bewegt, die in der Instruktion gespeichert ist. Die Bewegung wird mit den Einstellungen v2000 und z40 ausgeführt. Geschwindigkeit und Zonengröße des TCP betragen 2200 mm/s bzw. 45 mm.

Beispiel 2:

```
MoveAbsJ p5, v2000, fine \Inpos := inpos50, grip3;
```

Das Werkzeug grip3 wird entlang einer nicht linearen Bahn an eine absolute Achsenposition p5 bewegt. Der Roboter betrachtet die Position als erreicht, wenn 50 % der Positionsbedingung und 50 % der Geschwindigkeitsbedingung für einen fine-Stopppunkt erfüllt sind. Er wartet maximal zwei Sekunden auf die Erfüllung der Bedingungen. (siehe vordefinierte Daten inpos50 des Datentyps stoppointdata).

Beispiel 3

```
MoveAbsJ \Conc, *, v2000, z40, grip3;
```

Das Werkzeug grip3 wird entlang einer nicht linearen Bahn an eine absolute Achsenposition bewegt, die in der Instruktion gespeichert ist. Nachfolgende logische Instruktionen werden abgearbeitet, während sich der Roboter bewegt.

Beispiel 4:

```
MoveAbsJ \Conc, * \NoEOffs, v2000, z40, grip3;
```

Dieselbe Bewegung wie oben, jedoch wird die Bewegung nicht durch aktive Offsets für externe Achsen beeinflusst.

Beispiel 5:

```
GripLoad obj_mass;
MoveAbsJ start, v2000, z40, grip3 \WObj:= obj;
```

Der Roboter bewegt das Werkobjekt obj in Relation zum festen Werkzeug grip3 entlang einer nicht linearen Bahn an die absolute Achsenposition start.

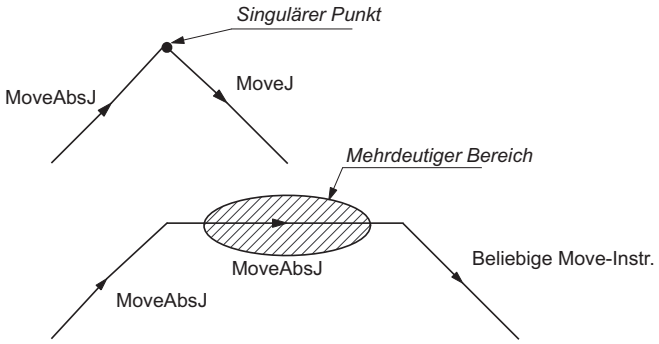
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Damit eine Rückwärtsabarbeitung mit der Instruktion `MoveAbsJ` möglich ist und Probleme mit singulären Punkten oder mehrdeutigen Bereichen vermieden werden, ist es unerlässlich, dass die nachfolgenden Instruktionen die folgenden Anforderungen erfüllen (siehe Abbildung unten).

Die Abbildung zeigt die Einschränkung für die Rückwärtsabarbeitung mit `MoveAbsJ`.



xx0500002201

`MoveAbsJ` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveAbsJ
  ['\' Conc ',']
  [ToJointPos ':='] <expression (IN) of jointtarget>
  ['\' ID ':=' <expression (IN) of identno>]
  ['\' NoEoffs]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' V ':=' <expression (IN) of num>]
  |['\' T ':=' <expression (IN) of num>]','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Z ':='] <expression (IN) of num >
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition von jointtarget	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.135 MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition

RobotWare - OS

Fortsetzung

Information über	Siehe
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Simultane Abarbeitung von Programmen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.136 MoveC - Kreisförmige Bewegung des Roboters

Verwendung

`MoveC` wird verwendet, um den Werkzeugarbeitspunkt (TCP) kreisförmig an ein vorgegebenes Ziel zu bewegen. Während der Bewegung bleibt die Orientierung gewöhnlich in Relation zum Kreisbogen unverändert.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveC`:

Siehe auch *Weitere Beispiele auf Seite 394*.

Beispiel 1:

```
MoveC p1, p2, v500, z30, tool2;
```

Der TCP von Werkzeug `tool2` wird mit den Geschwindigkeitsdaten `v500` und den Zonendaten `z30` kreisförmig an die Position `p2` bewegt. Der Kreisbogen wird durch die Startposition, den Kreispunkt `p1` und den Zielpunkt `p2` definiert.

Beispiel 2:

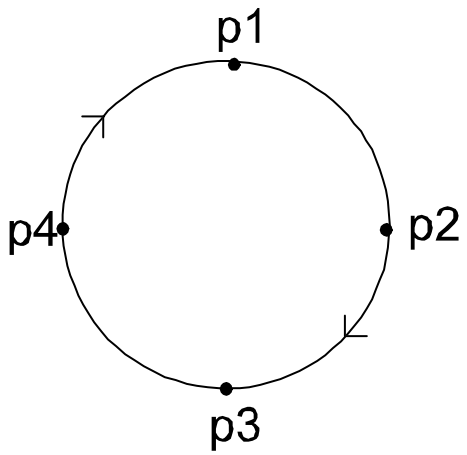
```
MoveC *, *, v500 \T:=5, fine, grip3;
```

Der TCP von Werkzeug `grip3` wird kreisförmig an einen Feinpunkt bewegt, der in der Instruktion gespeichert ist (gekennzeichnet durch das zweite *). Der Kreispunkt wird ebenfalls in der Instruktion gespeichert (gekennzeichnet durch das erste *). Die vollständige Bewegung dauert `5` Sekunden.

Beispiel 3

```
MoveL p1, v500, fine, tool1;
MoveC p2, p3, v500, z20, tool1;
MoveC p4, p1, v500, fine, tool1;
```

Die Abbildung zeigt, wie ein vollständiger Kreis durch zwei `MoveC` -Instruktionen ausgeführt wird .



xx0500002212

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.136 MoveC - Kreisförmige Bewegung des Roboters
RobotWare - OS
Fortsetzung

Argumente

```
MoveC [\Conc] CirPoint ToPoint [\ID] Speed [\V] | [\T] Zone [\Z]
     [\Inpos] Tool [\WObj] [\Corr] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp:`switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Normalerweise wird das Argument nicht verwendet. Es kann jedoch bei der Verwendung von Fly-By-Punkten eingesetzt werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden. Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`CirPoint`

Datentyp:`robtarget`

Der Bogenpunkt des Roboters: Der Kreispunkt ist eine Position auf dem Kreisbogen zwischen dem Startpunkt und dem Zielpunkt. Für größte Genauigkeit sollte er ungefähr in der Mitte zwischen Start- und Zielpunkt platziert werden. Wenn er zu nahe an den Start- oder Zielpunkt gesetzt wird, gibt das Robotersystem evtl. eine Warnung aus. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). Die Position der externen Achsen wird nicht verwendet.

`ToPoint`

Datentyp:`robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des TCP, der Werkzeugumorientierung und der externen Achsen.

`[ \V ]`

Velocity

Datentyp: `num`

Dieses Argument wird verwendet, um die Geschwindigkeit des TCP in mm/s direkt in der Instruktion zu definieren. Es ersetzt dann die entsprechende Geschwindigkeit, die in den Geschwindigkeitsdaten angegeben ist.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp:`zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Z ]`

Zone

Datentyp: `num`

Dieses Argument wird verwendet, um die Position des Roboter-TCP direkt in der Instruktion anzugeben. Die Länge der Zonenbahn wird in mm angegeben und ersetzt dann die entsprechende Zone in den Zonendaten.

`[ \Inpos ]`

In position

Datentyp:`stoppoint data`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.136 MoveC - Kreisförmige Bewegung des Roboters
RobotWare - OS
Fortsetzung

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Objekt-Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine kreisförmige Bahn in Relation zum Werkobjekt möglich ist.

`[ \Corr ]`

Correction

Datentyp: `switch`

Korrekturdaten, die durch die Instruktion `CorrWrite` in einen Korrektureintrag geschrieben wurden, werden der Bahn und der Zielposition hinzugefügt, falls dieses Argument vorhanden ist.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

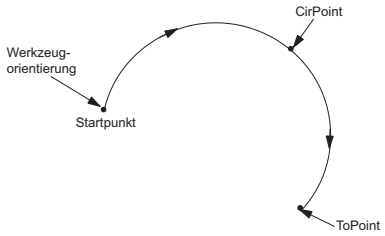
Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Der Roboter und die externen Einheiten werden wie folgt an den Zielpunkt bewegt:

- Der TCP des Werkzeugs wird kreisförmig bei konstanter programmierter Geschwindigkeit bewegt.
- Das Werkzeug wird bei konstanter Geschwindigkeit von der Orientierung an der Startposition zur Orientierung am Zielpunkt umorientiert.
- Die Umorientierung erfolgt relativ zur kreisförmigen Bahn. Wenn also die Orientierung in Relation zur Bahn am Start- und Zielpunkt identisch ist, bleibt die relative Orientierung während der Bewegung unverändert (siehe Abbildung unten).

Die Abbildung zeigt die Werkzeugorientierung bei kreisförmiger Bewegung.



xx0500002214

Die Orientierung am Kreispunkt wird nicht erreicht. Sie dient nur der Unterscheidung zwischen zwei möglichen Richtungen der Umorientierung. Die Genauigkeit der Umorientierung entlang der Bahn hängt nur von der Orientierung an Start- und Zielpunkt ab.

Unterschiedliche Modi für die Werkzeugorientierung auf der kreisförmigen Bahn sind in Instruktion `CirPathMode` beschrieben.

Unkoordinierte externe Achsen werden in konstanter Geschwindigkeit bewegt, damit sie den Zielpunkt gleichzeitig mit den Roboterachsen erreichen. Die Position am Kreispunkt wird nicht verwendet.

Wenn es nicht möglich ist, die programmierte Geschwindigkeit für die Umorientierung von externen Achsen beizubehalten, wird die Geschwindigkeit des TCP reduziert.

Eine Zonenbahn wird gewöhnlich generiert, wenn Bewegung an den nächsten Bahnabschnitt übertragen wird. Falls in den Zonendaten ein Stopppunkt angegeben ist, fährt die Programmabarbeitung nur fort, wenn der Roboter und die externen Achsen die programmierte Position erreicht haben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.136 MoveC - Kreisförmige Bewegung des Roboters
RobotWare - OS
Fortsetzung

Wenn der Startpunkt, der Kreispunkt und der Zielpunkt kollinear sind, führt die `MoveC`-Instruktion zu einer Linearbewegung.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_CONC_MAX`	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument `\Conc` verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `MoveC` sind unten aufgeführt.

Beispiel 1:

```
MoveC *, *, v500 \V:=550, z40 \Z:=45, grip3;
```

Der TCP von Werkzeug `grip3` wird kreisförmig an eine Position bewegt, die in der Instruktion gespeichert ist. Die Bewegung erfolgt mit den Einstellungen der Daten auf `v500` und `z40`; die Geschwindigkeit und Zonengröße des TCP betragen `550` mm/s bzw. `45` mm.

Beispiel 2:

```
MoveC p5, p6, v2000, fine \Inpos := inpos50, grip3;
```

Der TCP von Werkzeug `grip3` wird kreisförmig an den Stopppunkt `p6` bewegt. Der Roboter betrachtet die Position als erreicht, wenn 50 % der Positionsbedingung und 50 % der Geschwindigkeitsbedingung für einen `fine`-Stopppunkt erfüllt sind. Er wartet maximal zwei Sekunden auf die Erfüllung der Bedingungen. Siehe vordefinierte Daten `inpos50` vom Datentyp `stoppointdata`.

Beispiel 3

```
MoveC \Conc, *, *, v500, z40, grip3;
```

Der TCP von Werkzeug `grip3` wird kreisförmig an eine Position bewegt, die in der Instruktion gespeichert ist. Der Kreispunkt wird ebenfalls in der Instruktion gespeichert. Nachfolgende logische Instruktionen werden abgearbeitet, während sich der Roboter bewegt.

Beispiel 4:

```
MoveC cir1, p15, v500, z40, grip3 \WObj:=fixture;
```

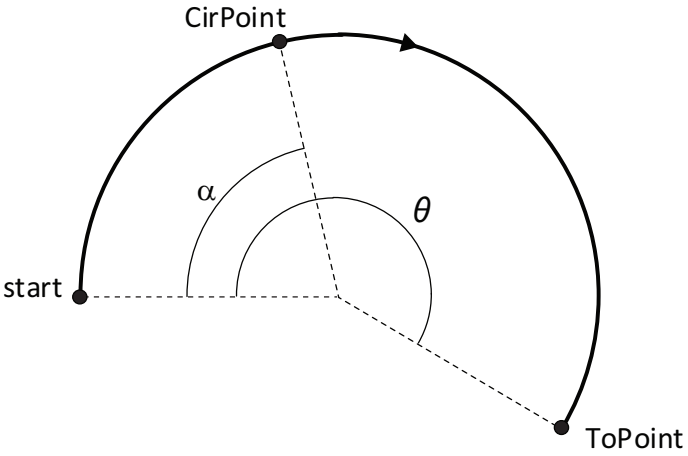
Der TCP von Werkzeug `grip3` wird über den Kreispunkt `cir1` kreisförmig an die Position `p15` bewegt. Diese Positionen werden im Objekt-Koordinatensystem für `fixture` angegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Es gibt einige Einschränkungen bei der Platzierung von `CirPoint` und `ToPoint`.



xx1700001575

- Mindestabstand zwischen Start und `ToPoint` beträgt 0,1 mm.
- Mindestabstand zwischen Start und `CirPoint` beträgt 0,1 mm.
- Der Mindestabstand zwischen `CirPoint` und `ToPoint` beträgt 0,1 mm.
- Wenn der Systemparameter *Restrict placing of circle points* auf Yes gesetzt ist, dann sind die folgenden zusätzlichen Einschränkungen aktiv:
  - Der Winkel der Kreisbahn (θ in der Abbildung oben) darf nicht größer als 240° sein.
  - Der Kreispunkt muss sich im mittleren Teil der Kreisbahn befinden (α muss gemäß dem obigen Bild 25-75 % von θ betragen).

Die Genauigkeit an den Grenzen kann gering sein; wenn z. B. Startpunkt und `ToPoint` auf dem Kreisbogen nahe beieinander liegen, ist der Fehler durch die Neigung des Kreises viel größer als die Genauigkeit, mit der die Punkte programmiert wurden.

Stellen Sie sicher, dass der Roboter während der Programmabarbeitung den Kreispunkt erreichen und bei Bedarf die Reihenfolge der Kreisbewegung aufteilen kann.

Eine Änderung des Abarbeitungsmodus von vorwärts in rückwärts oder umgekehrt, während der Roboter auf einer kreisförmigen Bahn stoppt, ist nicht zulässig und verursacht eine Fehlermeldung.

WARNUNG

Die Instruktion `MoveC` (oder jede andere Instruktion mit kreisförmiger Bewegung) darf nie von Beginn an neu gestartet werden, wenn sich der TCP zwischen dem Kreispunkt und dem Endpunkt befindet. In diesem Fall arbeitet der Roboter die programmierte Bahn nicht weiter ab (die Positionierung an der kreisförmigen Bahn erfolgt nicht in der programmierten Richtung).

Um das Risiko zu minimieren, setzen Sie den Systemparameter *Restrict placing of circlepoints* auf Yes (Typ *Motion Planner*, Thema *Motion*).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.136 MoveC - Kreisförmige Bewegung des Roboters

RobotWare - OS

Fortsetzung

`MoveC` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveC
  ['\' Conc ',']
  [CirPoint ':='] <expression (IN) of robtarget>','
  [ToPoint ':='] <expression (IN) of robtarget>','
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' V ':=' <expression (IN) of num>]
  |['\' T ':=' <expression (IN) of num>]','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Z ':=' <expression (IN) of num>]
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]
  ['\' Corr]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Werkzeugumorientierung auf kreisförmiger Bahn	*CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn auf Seite 133*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Simultane Abarbeitung von Programmen	*Technisches Referenzhandbuch - RAPID Overview*
Systemparameter	*Technisches Referenzhandbuch - Systemparameter*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.137 MoveCAO - Für eine kreisförmige Bewegung des Roboters und der Festlegung eines analogen Ausgangs in der Ecke

Verwendung

`MoveCAO` (*Move Circular Analog Output*) wird verwendet um den Werkzeugarbeitspunkt (TCP) kreisförmig in Richtung eines vorgegebenen Ziels zu bewegen. Der spezifizierte analoge Ausgang ist in der Mitte der Zonenbahn am Zielpunkt festgelegt. Während der Bewegung bleibt die Orientierung in Relation zum Kreisbogen normalerweise unverändert.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveCAO`:

Beispiel 1:

```
MoveCAO p1, p2, v500, z30, tool2, ao1, 1.1;
```

Der TCP von Werkzeug `tool2` wird mit den Geschwindigkeitsdaten `v500` und den Zonendaten `z30` kreisförmig an die Position `p2` bewegt. Der Kreisbogen wird durch die Startposition, den Kreispunkt `p1` und den Zielpunkt `p2` definiert. Ausgang `ao1` wird in der Mitte der Zonenbahn an `p2` gesetzt.

Argumente

```
MoveCAO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal
    Value [\TLoad]
```

`CirPoint`

Datentyp: `robtarget`

Der Bogenpunkt des Roboters: Der Kreispunkt ist eine Position auf dem Kreisbogen zwischen dem Startpunkt und dem Zielpunkt. Für größte Genauigkeit sollte er ungefähr in der Mitte zwischen Start- und Zielpunkt platziert werden. Wenn er zu nahe am Start- oder Zielpunkt gesetzt wird, gibt das Robotersystem evtl. eine Warnung aus. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). Die Position der externen Achsen wird nicht verwendet.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des TCP, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Objekt-Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine kreisförmige Bahn in Relation zum Werkobjekt möglich ist.

`Signal`

Datentyp: `signalao`

Der Name des analogen Ausgangssignals, das geändert werden soll.

`Value`

Datentyp: `num`

Der gewünschte Signalwert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.137 MoveCAO - Für eine kreisförmige Bewegung des Roboters und der Festlegung eines analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

```
[ \TLoad ]
```

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Siehe Instruktion `MoveC` für weitere Informationen über kreisförmige Bewegung, *MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*.

Das analoge Ausgangssignal befindet sich in der Mitte der Zonenbahn für Fly-By-Punkte, wie die untere Abbildung zeigt.

Die Abbildung zeigt das Festlegen eines analogen Ausgangssignals in der Zonenbahn mithilfe von `MoveCAO`.

```
MoveCAO p2, p2, v500, z30, tool2, ao1, 1.1;
```

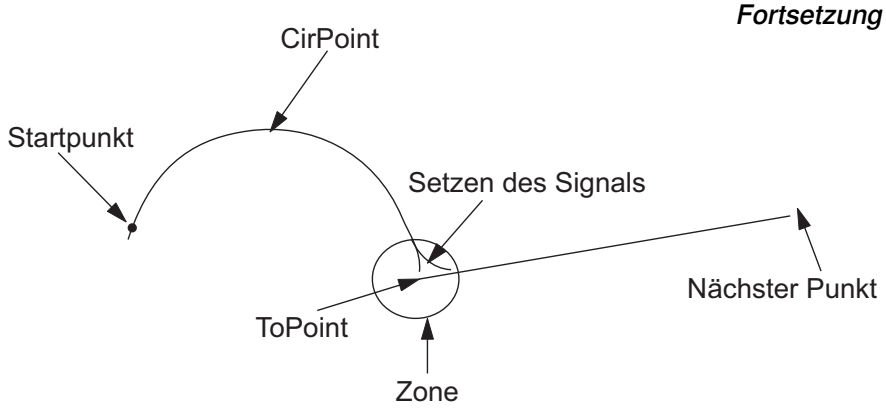
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.137 MoveCAO - Für eine kreisförmige Bewegung des Roboters und der Festlegung eines analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung



xx1400001116

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveC` und `SetAO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveCAO` verwendet wird, setzt das analoge Ausgangssignal ein, sobald der Roboter den Stopppunkt erreicht hat.

Das spezifizierte E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt, nicht jedoch, wenn es schrittweise rückwärts bewegt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AO_LIM`	Das programmierte `Value`-Argument für das angegebene analoge Ausgangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

Allgemeine Begrenzungen gemäß der Instruktion `MoveC`, siehe *MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*.

`MoveCAO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveCAO
  [CirPoint ':='] <expression (IN) of robtarget>','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.137 MoveCAO - Für eine kreisförmige Bewegung des Roboters und der Festlegung eines analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

```
[ToPoint ':='] <expression (IN) of robtarget>','
['\' ID ':=' <expression (IN) of identno>]','
[Speed ':='] <expression (IN) of speeddata>
|['\' T ':=' <expression (IN) of num>]','
[Zone ':='] <expression (IN) of zonedata>','
[Tool ':='] <persistent (PERS) of tooldata>
['\' WObj ':=' <persistent (PERS) of wobjdata>]','
[Signal ':='] <variable (VAR) of signalao>]','
[Value ':='] <expression (IN) of num>]
['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Kreisförmiges Bewegen des Roboters	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.138 MoveCDO – Der Roboter wird kreisförmig bewegt und es wird ein digitaler Ausgang in der Zonenbahn gesetzt

Verwendung

MoveCDO*Move Circular Digital Output*) wird verwendet, um den Werkzeugarbeitspunkt (TCP, Tool Center Point) kreisförmig an ein vorgegebenes Ziel zu bewegen. Der angegebene digitale Ausgang wird in der Mitte der Zonenbahn am Zielpunkt gesetzt/zurückgesetzt. Während der Bewegung bleibt die Orientierung gewöhnlich in Relation zum Kreisbogen unverändert.

Diese Instruktion kann nur in der Haupttask T_ROB1 oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion MoveCDO:

Beispiel 1:

```
MoveCDO p1, p2, v500, z30, tool2, do1,1;
```

Der TCP von Werkzeug tool2 wird mit den Geschwindigkeitsdaten v500 und den Zonendaten z30 kreisförmig an die Position p2 bewegt. Der Kreisbogen wird durch die Startposition, den Kreispunkt p1 und den Zielpunkt p2 definiert. Ausgang do1 wird in der Mitte der Zonenbahn an p2 gesetzt.

Argumente

```
MoveCDO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal
    Value [\TLoad]
```

CirPoint

Datentyp: robtarget

Der Bogenpunkt des Roboters: Der Kreispunkt ist eine Position auf dem Kreisbogen zwischen dem Startpunkt und dem Zielpunkt. Für größte Genauigkeit sollte er ungefähr in der Mitte zwischen Start- und Zielpunkt platziert werden. Wenn er zu nahe am Start- oder Zielpunkt gesetzt wird, gibt das Robotersystem evtl. eine Warnung aus. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). Die Position der externen Achsen wird nicht verwendet.

ToPoint

Datentyp: robtarget

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

[\ID]

Synchronization id

Datentyp: identno

Das Argument [\ID] ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.138 MoveCDO – Der Roboter wird kreisförmig bewegt und es wird ein digitaler Ausgang in der Zonenbahn gesetzt

RobotWare - OS

Fortsetzung

kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des TCP, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Objekt-Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine kreisförmige Bahn in Relation zum Werkobjekt möglich ist.

`Signal`

Datentyp: `signaldo`

Der Name des digitalen Ausgangssignals, das geändert werden soll.

`Value`

Datentyp: `dionum`

Der gewünschte Signalwert (0 oder 1).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.138 MoveCDO – Der Roboter wird kreisförmig bewegt und es wird ein digitaler Ausgang in der Zonenbahn gesetzt

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Siehe die Instruktion `MoveC` für Informationen über bogenförmige Bewegung.

Das digitale Ausgangsignal wird in der Mitte der Zonenbahn für Fly-By-Punkte gesetzt/zurückgesetzt, wie in der Abbildung unten gezeigt.

Fortsetzung auf nächster Seite

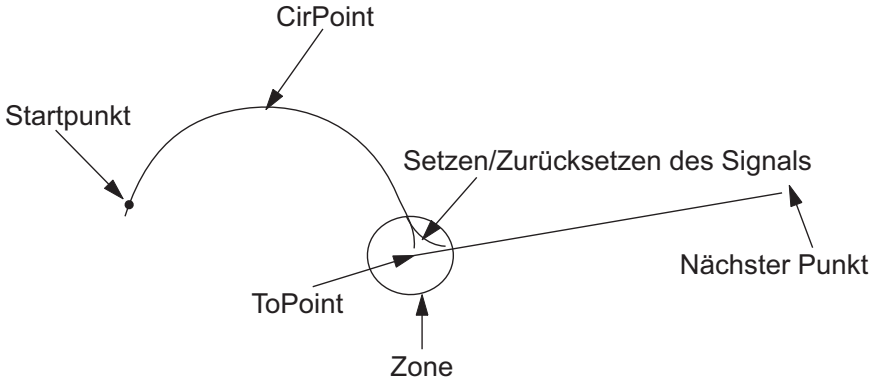
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.138 MoveCDO – Der Roboter wird kreisförmig bewegt und es wird ein digitaler Ausgang in der Zonenbahn gesetzt

RobotWare - OS

Fortsetzung

Die Abbildung zeigt das Setzen/Zurücksetzen des digitalen Ausgangssignals in der Zonenbahn mithilfe von `MoveCDO`.



xx0500002215

Für Stopppunkte wird die normale Programmiersequenz mit `MoveC` und `SetDO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveCDO` verwendet wird, wird das digitale Ausgangssignal festgelegt/zurückgesetzt, sobald der Roboter den Stopppunkt erreicht hat.

Das angegebene E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt/zurückgesetzt, aber nicht im Abarbeitungsmodus schrittweise rückwärts.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

Allgemeine Einschränkungen gemäß der Instruktion `MoveC`.

`MoveCDO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveCDO
  [CirPoint ':='] <expression (IN) of robtarget>','
  [ToPoint ':='] <expression (IN) of robtarget>','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.138 MoveCDO – Der Roboter wird kreisförmig bewegt und es wird ein digitaler Ausgang in der Zonenbahn gesetzt

RobotWare - OS

Fortsetzung

```
['\' ID ':=' <expression (IN) of identno>]','
[Speed ':='] <expression (IN) of speeddata>
|['\' T ':=' <expression (IN) of num>]','
[Zone ':='] <expression (IN) of zonedata>','
[Tool ':='] <persistent (PERS) of tooldata>
['\' WObj ':=' <persistent (PERS) of wobjdata>]','
[Signal ':='] <variable (VAR) of signaldo>]','
[Value ':='] <expression (IN) of dionum>]
['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Kreisförmiges Bewegen des Roboters	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.139 MoveCGO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des Gruppenausgangssignals in der Ecke

Verwendung

`MoveCGO` (*Move Circular Group Output*) wird verwendet um den Werkzeugarbeitspunkt (TCP) kreisförmig in Richtung eines vorgegebenen Ziels zu bewegen. Der spezifizierte analoge Ausgang wird in der Mitte der Zonenbahn am Zielpunkt festgelegt. Während der Bewegung bleibt die Orientierung in Relation zum Kreisbogen normalerweise unverändert.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveCGO`:

Beispiel 1:

```
MoveCGO p1, p2, v500, z30, tool2, go1 \Value:=5;
```

Das Werkzeug des TCP `tool2` wird kreisförmig an die Position `p2` mit der Datengeschwindigkeit `v500` und den Zonendaten `z30` bewegt. Der Kreis wird von einem Startpunkt, Kreispunkt `p1` und Zielpunkt `p2` definiert. Der Gruppenausgang `go1` wird in der Mitte der Zonenbahn auf `p2` festgelegt.

Argumente

```
MoveCGO CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal
    [\Value] | [\DValue] [\TLoad]
```

`CirPoint`

Datentyp: `robtarget`

Der Bogenpunkt des Roboters: Der Kreispunkt ist eine Position auf dem Kreisbogen zwischen dem Startpunkt und dem Zielpunkt. Für größte Genauigkeit sollte er ungefähr in der Mitte zwischen Start- und Zielpunkt platziert werden. Wenn er zu nahe am Start- oder Zielpunkt gesetzt wird, gibt das Robotersystem evtl. eine Warnung aus. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). Die Position der externen Achsen wird nicht verwendet.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des TCP, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Objekt-Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine kreisförmige Bahn in Relation zum Werkobjekt möglich ist.

`Signal`

Datentyp: `signalgo`

Der Name des Gruppenausgangs, der geändert werden soll.

`[ \Value ]`

Datentyp: `num`

Der gewünschte Signalwert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.139 MoveCGO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des Gruppenausgangssignals in der Ecke

RobotWare - OS

Fortsetzung

`[ \DValue ]`

Datentyp: `dnum`

Der gewünschte Signalwert.

Wenn keines der Argumente `\Value` oder `\DValue` eingegeben wurde, wird eine Fehlermeldung angezeigt.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

> **Hinweis**
>
> Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Siehe Instruktion `MoveC` für weitere Informationen über kreisförmige Bewegung, *MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*.

Der Gruppenausgang ist in der Mitte der Zonenbahn für Fly-By-Punkte festgelegt, wie die untere Abbildung zeigt.

Die Abbildung zeigt das Festlegen eines Gruppenausgangssignals in der Zonenbahn mithilfe von `MoveCGO`.

```
MoveCGO p2, p2, v500, z30, tool2, go1 \Value:=5;
```

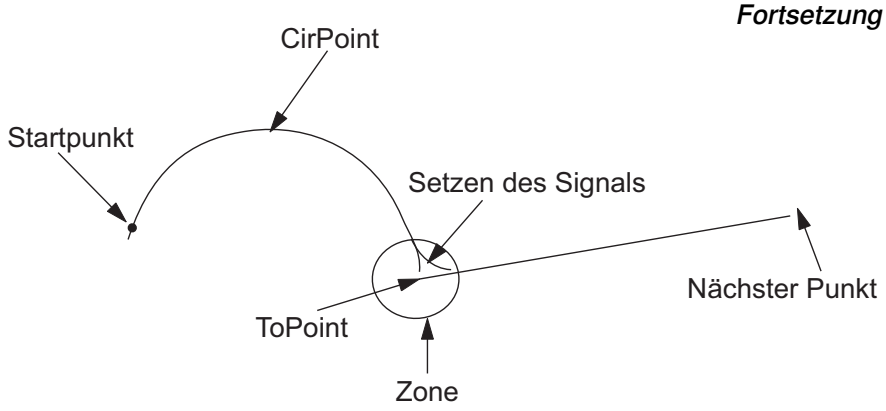
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.139 MoveCGO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des Gruppenausgangssignals in der Ecke

RobotWare - OS

Fortsetzung



xx1400001116

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveC` und `SetGO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveCGO` verwendet wird, wird der Gruppenausgang gesetzt, sobald der Roboter den Stopppunkt erreicht hat.

Das spezifizierte E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt, nicht jedoch, wenn es schrittweise rückwärts bewegt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_GO_LIM`	Das `Value` oder `DValue`-Argument für den spezifizierten Gruppenausgang liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

Allgemeine Begrenzungen gemäß der Instruktion `MoveC`, siehe *MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*.

`MoveCGO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveCGO
  [CirPoint ':='] <expression (IN) of robtarget>','
  [ToPoint ':='] <expression (IN) of robtarget>','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.139 MoveCGO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des Gruppenausgangssignals in der Ecke

RobotWare - OS

Fortsetzung

```
['\' ID ':=' <expression (IN) of identno>]','
[Speed ':='] <expression (IN) of speeddata>
|['\' T ':=' <expression (IN) of num>]','
[Zone ':='] <expression (IN) of zonedata>','
[Tool ':='] <persistent (PERS) of tooldata>
['\' WObj ':=' <persistent (PERS) of wobjdata>]','
[Signal ':='] <variable (VAR) of signalgo>]','
['\' Value ':='] <expression (IN) of num>]
|['\' Dvalue ':='] <expression (IN) of dnum>
['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Kreisförmiges Bewegen des Roboters	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.140 MoveCSync - Kreisförmiges Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur

Verwendung

`MoveCSync` (*Move Circular Synchronously*) wird verwendet, um den Werkzeugarbeitspunkt (TCP) kreisförmig an ein vorgegebenes Ziel zu bewegen. Die angegebene RAPID-Prozedur wird in der Mitte der Zonenbahn am Zielpunkt abgearbeitet. Während der Bewegung bleibt die Orientierung gewöhnlich in Relation zum Kreisbogen unverändert.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveCSync`:

Beispiel 1:

```
MoveCSync p1, p2, v500, z30, tool2, "proc1";
```

Der TCP von Werkzeug `tool2` wird mit den Geschwindigkeitsdaten `v500` und den Zonendaten `z30` kreisförmig an die Position `p2` bewegt. Der Kreisbogen wird durch die Startposition, den Kreispunkt `p1` und den Zielpunkt `p2` definiert. Ausgang `proc1` wird in der Mitte der Zonenbahn an `p2` ausgeführt.

Beispiel 2:

```
MoveCSync p1, p2, v500, z30, tool2, "MyModule:proc1";
```

Wie in Beispiel 1 oben, jedoch wird hier in der Mitte der Zonenbahn die lokal deklarierte Prozedur `proc1` im Modul `MyModule` aufgerufen.

Argumente

```
MoveCSync CirPoint ToPoint [\ID] Speed [\T] Zone Tool [\WObj]
    ProcName [\TLoad]
```

`CirPoint`

Datentyp: `robtarget`

Der Bogenpunkt des Roboters: Der Kreispunkt ist eine Position auf dem Kreisbogen zwischen dem Startpunkt und dem Zielpunkt. Für größte Genauigkeit sollte er ungefähr in der Mitte zwischen Start- und Zielpunkt platziert werden. Wenn er zu nahe am Start- oder Zielpunkt gesetzt wird, gibt das Robotersystem evtl. eine Warnung aus. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). Die Position der externen Achsen wird nicht verwendet.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.140 MoveCSync - Kreisförmiges Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur
RobotWare - OS
Fortsetzung

Datentyp: identno

Das Argument [\ID] ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

Speed

Datentyp: speeddata

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des TCP, der Werkzeugumorientierung und der externen Achsen.

[\T]

Time

Datentyp: num

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Zone

Datentyp: zonedata

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

Tool

Datentyp: tooldata

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

[\WObj]

Work Object

Datentyp: wobjdata

Das Werkobjekt (Objekt-Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

ProcName

Procedure Name

Datentyp: string

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name der RAPID-Prozedur, die in der Mitte der Zonenbahn am Zielpunkt abgearbeitet werden soll.

Die Prozedur wird auf TRAP-Ebene abgearbeitet (siehe Programmabarbeitung unten).

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Weitere Informationen über kreisförmige Bewegungen finden Sie bei den Erläuterungen zur Instruktion `MoveC` .

Die angegebene RAPID-Prozedur wird abgearbeitet, wenn der TCP die Mitte der Zonenbahn am Zielpunkt der Instruktion `MoveCSync` erreicht, wie in der Abbildung unten gezeigt.

Fortsetzung auf nächster Seite

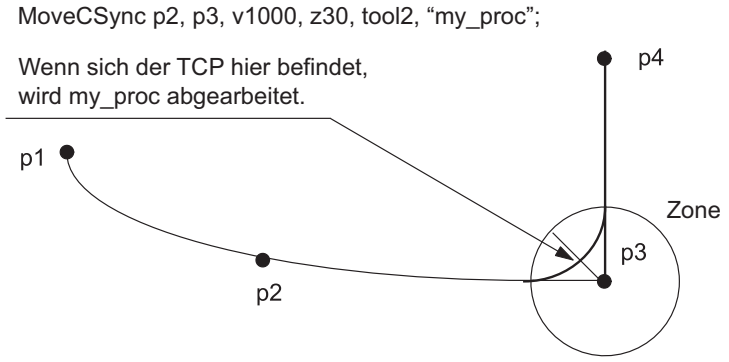
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.140 MoveCSync - Kreisförmiges Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur

RobotWare - OS

Fortsetzung

Die Abbildung zeigt, dass der Befehl zur Abarbeitung einer anwenderdefinierten RAPID-Prozedur in der Mitte der Zonenbahn erfolgt.



xx0500002216

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveC` und anderen RAPID-Instruktionen in Folge empfohlen.

Die Tabelle beschreibt die Abarbeitung der angegebenen RAPID-Prozedur in verschiedenen Modi.

Abarbeitungsmodus	Abarbeitung einer RAPID-Prozedur
Kontinuierlich oder zyklisch	Gemäß dieser Beschreibung
Vorwärtsschritt	Im Stopppunkt
Rückwärtsschritt	Überhaupt nicht

`MoveCSync` ist eine Einkapselung der Instruktionen `TriggInt` und `TriggC`. Der Prozeduraufruf wird auf TRAP-Ebene abgearbeitet.

Wenn die Mitte der Zonenbahn im Zielpunkt während der Verlangsamung nach einem Programmstopp erreicht wird, wird die Prozedur nicht aufgerufen (Programmabarbeitung wird gestoppt). Der Prozeduraufruf wird beim nächsten Programmstart ausgeführt.

Einschränkungen

Allgemeine Einschränkungen gemäß der Instruktion `MoveC`.

Wenn der Roboter die Mitte der Zonenbahn erreicht, wird die RAPID-Routine in der Regel nach einer Verzögerung von 20-30 ms abgearbeitet, je nach Typ der zu diesem Zeitpunkt ausgeführten Bewegung.

Ein Wechsel des Abarbeitungsmodus nach einem Programmstopp von kontinuierlich oder zyklisch zu vorwärts oder rückwärts ergibt einen Fehler. Dieser Fehler teilt dem Anwender mit, dass der Moduswechsel zum Auslassen der RAPID-Prozedur in der Warteschlange für die Abarbeitung auf der Bahn führen kann.

Die Instruktion `MoveCSync` kann nicht auf der Interrupt-Ebene verwendet werden. Die angegebene RAPID-Prozedur kann nicht mit einer schrittweisen Abarbeitung getestet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.140 MoveCSync - Kreisförmiges Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur

`MoveCSync` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveCSync
  [CirPoint ':='] <expression (IN) of robtarget>','
  [ToPoint ':='] <expression (IN) of robtarget>','
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  |['\' T ':=' <expression (IN) of num>]','
  [Zone ':='] <expression (IN) of zonedata>','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]','
  [ProcName ':='] <expression (IN) of string>]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Kreisförmige Bewegung des Roboters	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Definieren eines positionsabhängigen Interrupts	*TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Kreisförmige Roboterbewegung mit Ereignissen	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.141 MoveExtJ - Bewegen einer oder mehrerer mechanischer Einheiten ohne TCP

Verwendung

`MoveExtJ` (*Move External Joints*) wird verwendet, um externe lineare oder externe Rotationsachsen zu bewegen. Die externen Achsen können zu einer oder mehreren mechanischen Einheiten ohne TCP gehören.

Die Instruktion kann nur verwendet werden, wenn eine aktuelle Programmtask als Bewegungstask definiert ist und wenn diese eine oder mehrere mechanische Einheiten ohne TCP steuert.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveExtJ`:

Siehe auch *Weitere Beispiele auf Seite 420*.

Beispiel 1:

```
MoveExtJ jpos10, vrot10, z50;
```

Bewegen der externen Rotationsachsen an die Achsenposition `jpos10` mit der Geschwindigkeit `10` Grad/s und den Zonendaten `z50`.

Beispiel 2:

```
MoveExtJ \Conc, jpos20, vrot10 \T:=5, fine \InPos:=inpos20;
```

Bewegen der externen Achsen an die Achsenposition `jpos20` bei 5. Die Programmabarbeitung wird unverzüglich fortgesetzt, doch die externen Achsen werden an `jpos20` angehalten, bis die Konvergenzkriterien in `inpos20` erfüllt sind.

Argumente

```
MoveExtJ [\Conc] ToJointPos [\ID] [\UseEOffs] Speed [\T] Zone
     [\Inpos]
```

`[ \Conc ]`

Concurrent

Datentyp: `switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich die externe Achse bewegt. Normalerweise wird das Argument nicht verwendet. Es kann jedoch bei der Verwendung von Fly-By-Punkten eingesetzt werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden. Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn dieses Argument ausgelassen wird und der `ToJointPos` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor die externe Achse die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToJointPos`

To Joint Position

Datentyp: `jointtarget`

Der Zielpunkt als absolute Achsenposition der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization ID

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`[ \UseEOffs ]`

Use External Offset

Datentyp: `switch`

Der mit der Instruktion `EOffsSet` eingerichtete Offset für externe Achsen wird für die `MoveExtJ`-Instruktion aktiviert, wenn das Argument `UseEOffs` verwendet wird. Weitere Informationen über externe Offsets finden Sie in den Erläuterungen zur Instruktion `EOffsSet`.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Diese Daten definieren die Geschwindigkeit für die externen linearen und die externen Rotationsachsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Zone

Datentyp: zonedata

Zonendaten für die Bewegung. Die Zonendaten definieren einen Stopppunkt oder Fly-By-Punkt. Bei einem Fly-By-Punkt beschreibt die Zonengröße die Verzögerung und Beschleunigung der externen linearen Achsen oder externen Rotationsachsen.

[\Inpos]

In position

Datentyp: stoppointdata

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position der externen Achse im Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter Zone angegeben ist.

Programmabarbeitung

Die externen linearen Achsen oder die externen Rotationsachsen werden mit der programmierten Geschwindigkeit an den programmierten Punkt bewegt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_CONC_MAX	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument \Conc verwenden, wurde überschritten.

Weitere Beispiele

```
CONST jointtarget j1 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[0,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget j2 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[30,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget j3 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[60,9E9,9E9,9E9,9E9,9E9]];
CONST jointtarget j4 :=
     [[9E9,9E9,9E9,9E9,9E9,9E9],[90,9E9,9E9,9E9,9E9,9E9]];
CONST speeddata rot_ax_speed := [0, 0, 0, 45];

MoveExtJ j1, rot_ax_speed, fine;
MoveExtJ j2, rot_ax_speed, z20;
MoveExtJ j3, rot_ax_speed, z20;
MoveExtJ j4, rot_ax_speed, fine;
```

In diesem Beispiel wird die einzelne Rotationsachse an die Achsenposition 0 bewegt, und zwar um 30, 60 und 90 Grad mit einer Geschwindigkeit von 45 Grad/s.

Einschränkungen

MoveExtJ kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.141 MoveExtJ - Bewegen einer oder mehrerer mechanischer Einheiten ohne TCP

RobotWare - OS

Fortsetzung

Syntax

```
MoveExtJ
  [ '\' Conc ',' ]
  [ ToJointPos ':=' ] < expression (IN) of jointtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ '\' UseEOffs ',' ]
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos ':=' < expression (IN) of stoppointdata >] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition von jointtarget	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Simultane Abarbeitung von Programmen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.142 MoveJ - Achsenbewegung des Roboters

Verwendung

Mit `MoveJ` wird der Roboter schnell von einem Punkt zu einem anderen bewegt, wenn diese Bewegung nicht auf einer Geraden erfolgen muss.

Der Roboter und die externen Achsen bewegen sich entlang einer nicht linearen Bahn an die Zielposition. Alle Achsen erreichen die Zielposition gleichzeitig.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveJ`:

Siehe auch *Weitere Beispiele auf Seite 425*.

Beispiel 1:

```
MoveJ p1, vmax, z30, tool2;
```

Der Werkzeugarbeitspunkt (TCP) von Werkzeug `tool2` wird entlang einer nicht linearen Bahn an die Position `p1` mit der Geschwindigkeit `vmax` und den Zonendaten `z30` bewegt.

Beispiel 2:

```
MoveJ *, vmax \T:=5, fine, grip3;
```

Der TCP von Werkzeug `grip3` wird entlang einer nicht linearen Bahn an einen Stopppunkt bewegt, der in der Instruktion gespeichert ist (gekennzeichnet durch ein *). Die vollständige Bewegung dauert `5` Sekunden.

Argumente

```
MoveJ [\Conc] ToPoint [\ID] Speed [\V] | [\T] Zone [\Z] [\Inpos]
     Tool [\WObj] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp: `switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Normalerweise wird das Argument nicht verwendet. Es kann jedoch bei der Verwendung von Fly-By-Punkten eingesetzt werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden. Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID]`

Synchronization id

Datentyp: `identno`

Dieses Argument muss in einem MultiMove-System bei koordinierter, synchronisierter Bewegung verwendet werden und darf in keinem anderen Fall eingesetzt werden.

Die angegebene ID-Nummer muss in allen kooperierenden Programmtasks identisch sein. Durch die ID-Nummer wird gewährleistet, dass die Bewegungen bei Abarbeitung nicht verwechselt werden.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \V ]`

Velocity

Datentyp: `num`

Dieses Argument wird verwendet, um die Geschwindigkeit des TCP in mm/s direkt in der Instruktion zu definieren. Es ersetzt dann die entsprechende Geschwindigkeit, die in den Geschwindigkeitsdaten angegeben ist.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.142 MoveJ - Achsenbewegung des Roboters
RobotWare - OS
Fortsetzung

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Z ]`

Zone

Datentyp: `num`

Dieses Argument wird verwendet, um die Position des Roboter-TCP direkt in der Instruktion anzugeben. Die Länge der Zonenbahn wird in mm angegeben und ersetzt dann die entsprechende Zone in den Zonendaten.

`[ \Inpos ]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Der Werkzeugarbeitspunkt wird mit Interpolation der Achsenwinkel an den Zielpunkt bewegt. Dies bedeutet, dass jede Achse mit einer konstanten Achsengeschwindigkeit bewegt wird und dass alle Achsen gleichzeitig den Zielpunkt erreichen, wodurch sich eine nicht lineare Bahn ergibt.

Allgemein ausgedrückt bewegt sich der TCP in der ungefähren programmierten Geschwindigkeit (unabhängig davon, ob die externen Achsen koordiniert sind). Das Werkzeug wird umorientiert und die externen Achsen werden gleichzeitig mit dem TCP bewegt. Wenn es nicht möglich ist, die programmierte Geschwindigkeit für die Umorientierung oder für die externen Achsen beizubehalten, wird die Geschwindigkeit des TCP reduziert.

Eine Zonenbahn wird gewöhnlich generiert, wenn Bewegung an den nächsten Bahnabschnitt übertragen wird. Falls in den Zonendaten ein Stopppunkt angegeben ist, fährt die Programmabarbeitung nur fort, wenn der Roboter und die externen Achsen die programmierte Position erreicht haben.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_CONC_MAX`	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument `\Conc` verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `MoveJ` sind unten aufgeführt.

Beispiel 1:

```
MoveJ *, v2000\V:=2200, z40 \Z:=45, grip3;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.142 MoveJ - Achsenbewegung des Roboters

RobotWare - OS

Fortsetzung

Der TCP von Werkzeug `grip3` wird entlang einer nicht linearen Bahn an eine Position bewegt, die in der Instruktion gespeichert ist (gekennzeichnet durch ein *). Die Bewegung erfolgt mit den Einstellungen der Daten auf `v2000` und `z40`; die Geschwindigkeit und Zonengröße des TCP betragen `2200` mm/s bzw. `45` mm.

Beispiel 2:

```
MoveJ p5, v2000, fine \Inpos := inpos50, grip3;
```

Der TCP von Werkzeug `grip3` wird entlang einer nicht linearen Bahn an Stopppunkt `p5` bewegt. Der Roboter betrachtet die Position als erreicht, wenn 50 % der Positionsbedingung und 50 % der Geschwindigkeitsbedingung für einen `fine`-Stopppunkt erfüllt sind. Er wartet maximal zwei Sekunden auf die Erfüllung der Bedingungen Siehe vordefinierte Daten `inpos50` vom Datentyp `stoppointdata`.

Beispiel 3

```
MoveJ \Conc, *, v2000, z40, grip3;
```

Der TCP von Werkzeug `grip3` wird entlang einer nicht linearen Bahn an eine Position bewegt, die in der Instruktion gespeichert ist (gekennzeichnet durch ein *). Nachfolgende logische Instruktionen werden abgearbeitet, während sich der Roboter bewegt.

Beispiel 4:

```
MoveJ start, v2000, z40, grip3 \WObj:=fixture;
```

Der TCP von Werkzeug `grip3` wird entlang einer nicht linearen Bahn an die Position `start` bewegt. Diese Position wird im Objekt-Koordinatensystem für die Vorrichtung `fixture` angegeben.

Einschränkungen

`MoveJ` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveJ
  ['\' Conc ',' ]
  [ToPoint ':=' ] < expression (IN) of robtarget >
  ['\' ID ':=' < expression (IN) of identno >]','
  [Speed ':=' ] < expression (IN) of speeddata >
  ['\' V ':=' < expression (IN) of num > ]
  |['\' T ':=' <expression (IN) of num>]','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Z ':=' <expression (IN) of num>]
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Simultane Abarbeitung von Programmen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.143 MoveJAO - Für eine Achsenbewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

Verwendung

`MoveJAO` (*Move Joint Analog Output*) wird verwendet um den Roboter schnell von einem Punkt zum nächsten zu bewegen, wenn die Bewegung nicht linear verlaufen muss. Der spezifizierte analoge Ausgang ist in der Mitte der Zonenbahn festgelegt.

Der Roboter und die externen Achsen bewegen sich entlang einer nicht linearen Bahn an die Zielposition. Alle Achsen erreichen die Zielposition gleichzeitig.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveJAO`:

Beispiel 1:

```
MoveJAO p1, vmax, z30, tool2, ao1, 1.1;
```

Der Werkzeugarbeitspunkt (TCP) von Werkzeug `tool2` wird entlang einer nicht linearen Bahn an die Position `p1` mit der Geschwindigkeit `vmax` und den Zonendaten `z30` bewegt. Ausgang `ao1` wird in der Mitte der Zonenbahn an `p1` gesetzt.

Argumente

```
MoveJAO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`Signal`

Datentyp: `signalao`

Der Name des analogen Ausgangssignals, das geändert werden soll.

`Value`

Datentyp: `num`

Der gewünschte Signalwert.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.143 MoveJAO - Für eine Achsenbewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

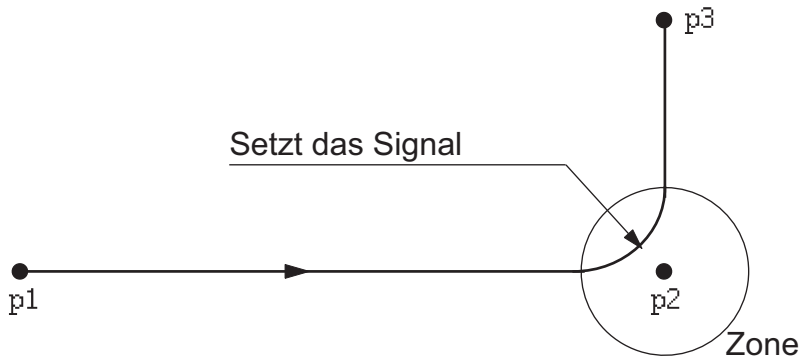
Programmabarbeitung

Siehe Instruktion `MoveJ` für weitere Informationen über die Achsenbewegung, *MoveJ - Achsenbewegung des Roboters auf Seite 422*.

Das analoge Ausgangssignal befindet sich in der Mitte der Zonenbahn für Fly-By-Punkte, wie die untere Abbildung zeigt.

Die Abbildung zeigt das Festlegen eines analogen Ausgangssignals in der Zonenbahn mithilfe von `MoveJAO`.

```
MoveJAO p2, vmax, z30, tool2, ao1, 1.1;
```



xx1400001118

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveJ` und `SetAO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveJAO` verwendet wird, setzt das analoge Ausgangssignal ein, sobald der Roboter den Stopppunkt erreicht hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.143 MoveJAO - Für eine Achsenbewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

Das spezifizierte E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt, nicht jedoch, wenn es schrittweise rückwärts bewegt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AO_LIM`	Das programmierte `Value`-Argument für das angegebene analoge Ausgangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

`MoveJAO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveJAO
  [ToPoint ':='] <expression (IN) of robtarget>
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  |['\' T ':=' <expression (IN) of num>]','
  [Zone ':='] <expression (IN) of zonedata>','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]','
  [Signal ':='] <variable (VAR) of signalao>]','
  [Value ':=' ] <expression (IN) of num>]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Achsenbewegung des Roboters	*MoveJ - Achsenbewegung des Roboters auf Seite 422*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.143 MoveJAO - Für eine Achsenbewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Information über	Siehe
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.144 MoveJDO - Achsenbewegung des Roboters und Setzen eines digitalen Ausgangs in der Zonenbahn

Verwendung

`MoveJDO` (*Move Joint Digital Output*) wird verwendet, um den Roboter rasch von einem Punkt an einen anderen zu bewegen, wenn diese Bewegung nicht geradlinig erfolgen muss. Das angegebene digitale Ausgangssignal wird in der Mitte der Zonenbahn gesetzt/zurückgesetzt.

Der Roboter und die externen Achsen bewegen sich entlang einer nicht linearen Bahn an die Zielposition. Alle Achsen erreichen die Zielposition gleichzeitig.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveJDO`:

Beispiel 1:

```
MoveJDO p1, vmax, z30, tool2, do1, 1;
```

Der Werkzeugarbeitspunkt (TCP) von Werkzeug `tool2` wird entlang einer nicht linearen Bahn an die Position `p1` mit der Geschwindigkeit `vmax` und den Zonendaten `z30` bewegt. Ausgang `do1` wird in der Mitte der Zonenbahn an `p1` gesetzt.

Argumente

```
MoveJDO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.144 MoveJDO - Achsenbewegung des Roboters und Setzen eines digitalen Ausgangs in der Zonenbahn
RobotWare - OS
Fortsetzung

Datentyp: num

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Zone

Datentyp: zonedata

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

Tool

Datentyp: tooldata

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

[\WObj]

Work Object

Datentyp: wobjdata

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

Signal

Datentyp: signaldo

Der Name des digitalen Ausgangssignals, das geändert werden soll.

Value

Datentyp: dionum

Der gewünschte Signalwert (0 oder 1).

[\TLoad]

Total load

Datentyp: loaddata

Das Argument \TLoad beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments \TLoad werden die loaddata in den aktuellen tooldata nicht berücksichtigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn das \TLoad-Argument auf load0 gesetzt ist, wird das \TLoad-Argument nicht berücksichtigt, und stattdessen werden die loaddata in den aktuellen tooldata verwendet.

Damit das \TLoad-Argument verwendet werden kann, muss der Wert des Systemparameters ModalPayLoadMode unbedingt auf 0 gesetzt werden. Wird ModalPayLoadMode auf 0 eingestellt, kann die GripLoad-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter ModalPayLoadMode auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die loaddata vom Werkzeug in eine bereits vorhandene oder eine neue persistente loaddata-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang SimMode (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die loaddata im optionalen Argument \TLoad nicht berücksichtigt und stattdessen die loaddata in den aktuellen tooldata verwendet.

Hinweis

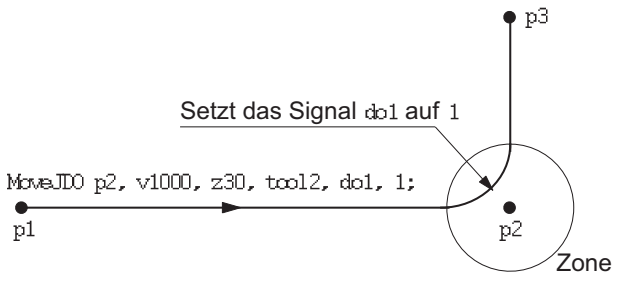
Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion GripLoad. Deshalb beträgt der Standardwert für den Systemparameter ModalPayLoadMode 1.

Programmabarbeitung

Weitere Informationen über Achsbewegungen finden Sie bei den Erläuterungen zur Instruktion MoveJ.

Das digitale Ausgangsignal wird in der Mitte der Zonenbahn für Fly-By-Punkte gesetzt/zurückgesetzt, wie in der Abbildung unten gezeigt.

Die Abbildung zeigt das Setzen/Zurücksetzen des digitalen Ausgangssignals in der Zonenbahn mithilfe von MoveJDO.



xx0500002196

Für Stopppunkte wird die normale Programmiersequenz mit MoveJ und SetDO empfohlen. Wenn jedoch der Stopppunkt in der Instruktion MoveJDO verwendet wird, wird das digitale Ausgangssignal festgelegt/zurückgesetzt, sobald der Roboter den Stopppunkt erreicht hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.144 MoveJDO - Achsenbewegung des Roboters und Setzen eines digitalen Ausgangs in der Zonenbahn

RobotWare - OS

Fortsetzung

Das angegebene E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt/zurückgesetzt, aber nicht im Abarbeitungsmodus schrittweise rückwärts.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

MoveJDO kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveJDO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata>
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ Signal ':=' ] < variable (VAR) of signaldo>] ','
  [ Value ':=' ] < expression (IN) of dionum > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Achsenbewegung des Roboters	*MoveJ - Achsenbewegung des Roboters auf Seite 422*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.144 MoveJDO - Achsenbewegung des Roboters und Setzen eines digitalen Ausgangs in der Zonenbahn

RobotWare - OS

Fortsetzung

Information über	Siehe
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.145 MoveJGO - Für eine Achsenbewegung des Roboters und Einstellen eines Gruppenausgangs in der Ecke

Verwendung

`MoveJGO` (*Move Joint Group Output*) wird verwendet um den Roboter schnell von einem Punkt zum nächsten zu bewegen, wenn die Bewegung nicht linear verlaufen muss. Das spezifizierte Gruppenausgangssignal ist in der Mitte der Zonenbahn festgelegt.

Der Roboter und die externen Achsen bewegen sich entlang einer nicht linearen Bahn an die Zielposition. Alle Achsen erreichen die Zielposition gleichzeitig.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveJGO`:

Beispiel 1:

```
MoveJGO p1, vmax, z30, tool2, go1 \Value:=5;
```

Der Werkzeugarbeitspunkt (TCP) des Werkzeugs `tool2` wird entlang einer nicht linearen Bahn an die Position `p1` mit der Geschwindigkeit `vmax` und den Zonendaten `z30` bewegt. Der Gruppenausgang `go1` ist in der Mitte der Zonenbahn auf `p1` festgelegt.

Argumente

```
MoveJGO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal [\Value]
     | [\DValue] [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.145 MoveJGO - Für eine Achsenbewegung des Roboters und Einstellen eines Gruppenausgangs in der Ecke

RobotWare - OS

Fortsetzung

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`Signal`

Datentyp: `signalgo`

Der Name des Gruppenausgangs, der geändert werden soll.

`[ \Value ]`

Datentyp: `num`

Der gewünschte Signalwert.

`[ \DValue ]`

Datentyp: `dnum`

Der gewünschte Signalwert.

Wenn keines der Argumente `\Value` oder `\DValue` eingegeben wurde, wird eine Fehlermeldung angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.145 MoveJGO - Für eine Achsenbewegung des Roboters und Einstellen eines Gruppenausgangs in der Ecke

RobotWare - OS

Fortsetzung

```
[ \TLoad ]
```

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Weitere Informationen über Achsbewegungen finden Sie bei den Erläuterungen zur Instruktion `MoveJ`.

Der Gruppenausgang ist in der Mitte der Zonenbahn für Fly-By-Punkte festgelegt, wie die untere Abbildung zeigt.

Die Abbildung zeigt das Festlegen eines Gruppenausgangssignals in der Zonenbahn mithilfe von `MoveJGO`.

```
MoveJGO p2, vmax, z30, tool2, go1 \Value:=5;
```

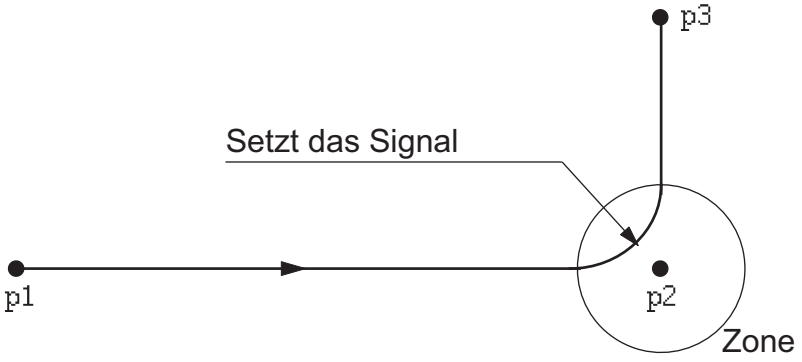
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.145 MoveJGO - Für eine Achsenbewegung des Roboters und Einstellen eines Gruppenausgangs in der Ecke

RobotWare - OS

Fortsetzung



xx1400001118

Für Stopppunkte wird die normale Programmiersequenz mit `MoveJ` und `SetGO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveJGO` verwendet wird, setzt der Gruppenausgang ein, sobald der Roboter den Stopppunkt erreicht hat.

Das spezifizierte E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt, nicht jedoch, wenn es schrittweise rückwärts bewegt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_GO_LIM`	Das `Value` oder `DValue`-Argument für den spezifizierten Gruppenausgang liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

`MoveJGO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveJGO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.145 MoveJGO - Für eine Achsenbewegung des Roboters und Einstellen eines Gruppenausgangs in der Ecke

RobotWare - OS

Fortsetzung

```
[ Tool ':=' ] < persistent (PERS) of tooldata>
[ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
[ Signal ':=' ] < variable (VAR) of signalgo>] ','
[ '\' Value ':=' ] < expression (IN) of num > ]
| [ '\' Dvalue' :=' ] < expression (IN) of dnum >
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Achsenbewegung des Roboters	*MoveJ - Achsenbewegung des Roboters auf Seite 422*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.146 MoveJSync - Bewegen des Roboters durch Achsbewegung und Abarbeiten einer RAPID-Prozedur

Verwendung

`MoveJSync`(*Move Joint Synchronously*) wird verwendet, um den Roboter rasch von einem Punkt an einen anderen zu bewegen, wenn diese Bewegung nicht geradlinig erfolgen muss. Die angegebene RAPID-Prozedur wird in der Mitte der Zonenbahn am Zielpunkt abgearbeitet.

Der Roboter und die externen Achsen bewegen sich entlang einer nicht linearen Bahn an die Zielposition. Alle Achsen erreichen die Zielposition gleichzeitig.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveJSync`:

Beispiel 1:

```
MoveJSync p1, vmax, z30, tool2, "proc1";
```

Der Werkzeugarbeitspunkt (TCP) von Werkzeug `tool2` wird entlang einer nicht linearen Bahn an die Position `p1` mit der Geschwindigkeit `vmax` und den Zonendaten `z30` bewegt. Ausgang `proc1` wird in der Mitte der Zonenbahn an `p1` ausgeführt.

Beispiel 2:

```
MoveJSync p1, vmax, z30, tool2, "MyModule:proc1";
```

Wie in Beispiel 1 oben, jedoch wird hier in der Mitte der Zonenbahn die lokal deklarierte Prozedur `proc1` im Modul `MyModule` aufgerufen.

Argumente

```
MoveJSync ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName
    [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`ProcName`

Procedure Name

Datentyp: `string`

Name der RAPID-Prozedur, die in der Mitte der Zonenbahn am Zielpunkt abgearbeitet werden soll. Der Prozeduraufruf ist ein Aufruf mit später Bindung, daher übernimmt er die Eigenschaften.

Die Prozedur wird auf TRAP-Ebene abgearbeitet (siehe Programmabarbeitung unten).

`[ \TLoad ]`

Total load

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

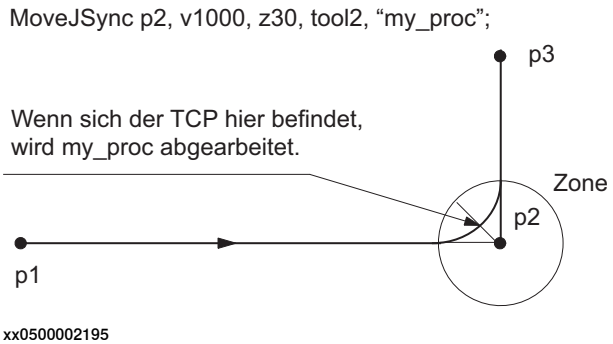
Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Weitere Informationen über Achsbewegungen finden Sie bei den Erläuterungen zur Instruktion `MoveJ`.

Die angegebene RAPID-Prozedur wird abgearbeitet, wenn der TCP die Mitte der Zonenbahn am Zielpunkt der Instruktion `MoveJSync` erreicht, wie in der Abbildung unten

gezeigt.



Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.146 MoveJSync - Bewegen des Roboters durch Achsbewegung und Abarbeiten einer RAPID-Prozedur

RobotWare - OS

Fortsetzung

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveJ` und anderen RAPID-Instruktionen in Folge empfohlen.

Die Tabelle beschreibt die Abarbeitung der angegebenen RAPID-Prozedur in verschiedenen Modi.

Abarbeitungsmodus	Abarbeitung einer RAPID-Prozedur
Kontinuierlich oder zyklisch	Gemäß dieser Beschreibung
Vorwärtsschritt	Im Stopppunkt
Rückwärtsschritt	Überhaupt nicht

`MoveJSync` ist eine Einkapselung der Instruktionen `TriggInt` und `TriggJ`. Der Prozeduraufruf wird auf TRAP-Ebene abgearbeitet.

Wenn die Mitte der Zonenbahn im Zielpunkt während der Verlangsamung nach einem Programmstopp erreicht wird, wird die Prozedur nicht aufgerufen (Programmabarbeitung wird gestoppt). Der Prozeduraufruf wird beim nächsten Programmstart ausgeführt.

Einschränkungen

Wenn der Roboter die Mitte der Zonenbahn erreicht, wird die `RAPID`-Routine in der Regel nach einer Verzögerung von 20-30 ms abgearbeitet, je nach Typ der zu diesem Zeitpunkt ausgeführten Bewegung.

Ein Wechsel des Abarbeitungsmodus nach einem Programmstopp von kontinuierlich oder zyklisch zu vorwärts oder rückwärts ergibt einen Fehler. Dieser Fehler teilt dem Anwender mit, dass der Moduswechsel zum Auslassen der RAPID-Prozedur in der Warteschlange für die Abarbeitung auf der Bahn führen kann.

Die Instruktion `MoveJSync` kann nicht auf der Interrupt-Ebene verwendet werden. Die angegebene RAPID-Prozedur kann nicht mit einer schrittweisen Abarbeitung getestet werden.

`MoveJSync` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveJSync
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj '=' < persistent (PERS) of wobjdata > ] ','
  [ ProcName '=' ] < expression (IN) of string > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Achsenbewegung des Roboters	*MoveJ - Achsenbewegung des Roboters auf Seite 422*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Definieren eines positionsabhängigen Interrupts	*TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Achsenweise Roboterbewegungen mit Ereignissen	*TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.147 MoveL - Lineare Bewegung des Roboters

Verwendung

`MoveL` wird verwendet, um den Werkzeugarbeitspunkt (TCP) linear an ein vorgegebenes Ziel zu bewegen. Wenn der TCP stationär bleiben soll, kann mit dieser Instruktion auch das Werkzeug umorientiert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` bzw. in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveL`:

Siehe auch *Weitere Beispiele auf Seite 452*.

Beispiel 1:

```
MoveL p1, v1000, z30, tool2;
```

Der TCP von Werkzeug `tool2` wird linear an die Position `p1` mit der Geschwindigkeit `v1000` und den Zonendaten `z30` bewegt.

Beispiel 2:

```
MoveL *, v1000\T:=5, fine, grip3;
```

Der TCP von Werkzeug `grip3` wird linear an einen Stopppunkt bewegt, der in der Instruktion gespeichert ist (gekennzeichnet durch ein *). Die vollständige Bewegung dauert `5` Sekunden.

Argumente

```
MoveL [\Conc] ToPoint [\ID] Speed [\V] | [ \T] Zone [\Z] [\Inpos]
     Tool [\WObj] [\Corr] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp: `switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Normalerweise wird das Argument nicht verwendet. Es kann jedoch bei der Verwendung von Fly-By-Punkten eingesetzt werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden. Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \V ]`

Velocity

Datentyp: `num`

Dieses Argument wird verwendet, um die Geschwindigkeit des TCP in mm/s direkt in der Instruktion zu definieren. Es ersetzt dann die entsprechende Geschwindigkeit, die in den Geschwindigkeitsdaten angegeben ist.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Z ]`

Zone

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.147 MoveL - Lineare Bewegung des Roboters
RobotWare - OS
Fortsetzung

Datentyp: num

Dieses Argument wird verwendet, um die Position des Roboter-TCP direkt in der Instruktion anzugeben. Die Länge der Zonenbahn wird in mm angegeben und ersetzt dann die entsprechende Zone in den Zonendaten.

[\Inpos]

In position

Datentyp: stoppointdata

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter Zone angegeben ist.

Tool

Datentyp: tooldata

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

[\WObj]

Work Object

Datentyp: wobjdata

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden. In diesem Fall ist die Position dem Weltkoordinatensystem zugeordnet. Wenn hingegen ein stationäres Werkzeug oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, um eine lineare Bewegung relativ zum Werkobjekt auszuführen.

[\Corr]

Correction

Datentyp: switch

Korrekturdaten, die durch die Instruktion CorrWrite in einen Korrektureintrag geschrieben wurden, werden der Bahn und der Zielposition hinzugefügt, falls dieses Argument vorhanden ist.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

[\TLoad]

Total load

Datentyp: loaddata

Das Argument \TLoad beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments \TLoad werden die loaddata in den aktuellen tooldata nicht berücksichtigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn das \TLoad-Argument auf load0 gesetzt ist, wird das \TLoad-Argument nicht berücksichtigt, und stattdessen werden die loaddata in den aktuellen tooldata verwendet.

Damit das \TLoad-Argument verwendet werden kann, muss der Wert des Systemparameters ModalPayLoadMode unbedingt auf 0 gesetzt werden. Wird ModalPayLoadMode auf 0 eingestellt, kann die GripLoad-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter ModalPayLoadMode auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die loaddata vom Werkzeug in eine bereits vorhandene oder eine neue persistente loaddata-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang SimMode (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die loaddata im optionalen Argument \TLoad nicht berücksichtigt und stattdessen die loaddata in den aktuellen tooldata verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion GripLoad. Deshalb beträgt der Standardwert für den Systemparameter ModalPayLoadMode 1.

Programmabarbeitung

Der Roboter und die externen Einheiten werden wie folgt an die Zielposition bewegt:

- Der TCP des Werkzeugs wird linear bei konstanter programmierter Geschwindigkeit bewegt.
- Das Werkzeug wird entlang der Bahn in gleichen Intervallen umorientiert.
- Unkoordinierte externe Achsen werden in konstanter Geschwindigkeit bewegt, damit sie den Zielpunkt gleichzeitig mit den Roboterachsen erreichen.

Wenn es nicht möglich ist, die programmierte Geschwindigkeit für die Umorientierung von externen Achsen beizubehalten, wird die Geschwindigkeit des TCP reduziert.

Eine Zonenbahn wird gewöhnlich generiert, wenn Bewegung an den nächsten Bahnabschnitt übertragen wird. Falls in den Zonendaten ein Stopppunkt angegeben ist, fährt die Programmabarbeitung nur fort, wenn der Roboter und die externen Achsen die programmierte Position erreicht haben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.147 MoveL - Lineare Bewegung des Roboters
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_CONC_MAX	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument \Conc verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion MoveL sind unten aufgeführt.

Beispiel 1:

```
MoveL *, v2000 \V:=2200, z40 \Z:=45, grip3;
```

Der TCP von Werkzeug grip3 wird linear an eine Position bewegt, die in der Instruktion gespeichert ist. Die Bewegung wird mit den Einstellungen v2000 und z40 ausgeführt. Geschwindigkeit und Zonengröße des TCP betragen 2200 mm/s bzw. 45 mm.

Beispiel 2:

```
MoveL p5, v2000, fine \Inpos := inpos50, grip3;
```

Der TCP von Werkzeug grip3 wird linear an den Stopppunkt p5 bewegt. Der Roboter betrachtet die Position als erreicht, wenn 50 % der Positionsbedingung und 50 % der Geschwindigkeitsbedingung für einen fine-Stopppunkt erfüllt sind. Er wartet maximal zwei Sekunden auf die Erfüllung der Bedingungen. Siehe vordefinierte Daten inpos50 vom Datentyp stoppointdata.

Beispiel 3

```
MoveL \Conc, *, v2000, z40, grip3;
```

Der TCP von Werkzeug grip3 wird linear an eine Position bewegt, die in der Instruktion gespeichert ist. Nachfolgende logische Instruktionen werden abgearbeitet, während sich der Roboter bewegt.

Beispiel 4:

```
MoveL start, v2000, z40, grip3 \WObj:=fixture;
```

Der TCP von Werkzeug grip3 wird linear an die Position start bewegt. Diese Position wird im Objekt-Koordinatensystem für die Vorrichtung fixture angegeben.

Beispiel mit TLoad

```
MoveL p1, v1000, fine, tool2;
! Pick up the payload
Set gripperdo;
MoveL p2, v1000, z30, tool2 \TLoad:=tool2piece;
MoveL p3, v1000, fine, tool2 \TLoad:=tool2piece;
! Release the payload
Reset gripperdo;
MoveL p4, v1000, fine, tool2;
```

Der TCP des Werkzeugs, tool2, wird linear an die Position p1 bewegt, um dort eine Nutzlast aufzunehmen. Von dieser Position aus wird der TCP mit der

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Gesamtlast `tool2piece` an die Positionen p2 und p3 bewegt. Die `loaddata` in den aktuellen `tooldata` werden nicht berücksichtigt. Die Nutzlast wird abgegeben und die Last des Werkzeugs erst wieder bei der Bewegung zu Position p4 berücksichtigt.

Einschränkungen

`MoveL` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveL
  [ '\' Conc ',' ]
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' V ':=' < expression (IN) of num > ]
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Z ':='< expression (IN) of num > ]
  [ '\' Inpos ':=' < expression (IN) of stoppointdata > ] ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ]
  [ '\' Corr ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Simultane Abarbeitung von Programmen	*Technisches Referenzhandbuch - RAPID Overview*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.147 MoveL - Lineare Bewegung des Roboters

RobotWare - OS

Fortsetzung

Information über	Siehe
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.148 MoveLAO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

Verwendung

`MoveLAO`(*Move Linearly Analog Output*) wird verwendet um den Werkzeugarbeitspunkt (TCP) linear in Richtung eines vorgegebenen Ziels zu bewegen. Der spezifizierte analoge Ausgang ist in der Mitte der Zonenbahn festgelegt.

Wenn der TCP stationär bleiben soll, kann mit dieser Instruktion auch das Werkzeug umorientiert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveLAO`:

Beispiel 1:

```
MoveLAO p1, v1000, z30, tool2, ao1, 1.1;
```

Der TCP des Werkzeugs `tool2` wird linear in Richtung der Position `p1` mit der Datengeschwindigkeit `v1000` und den Zonendaten `z30` bewegt. Der Gruppenausgang `ao1` ist in der Mitte der Zonenbahn am Zielpunkt `p1` festgelegt.

Argumente

```
MoveLAO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.148 MoveLAO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Datentyp: num

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Zone

Datentyp: zonedata

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

Tool

Datentyp: tooldata

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

[\WObj]

Work Object

Datentyp: wobjdata

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

Signal

Datentyp: signalao

Der Name des analogen Ausgangssignals, das geändert werden soll.

Value

Datentyp: num

Der gewünschte Signalwert.

[\TLoad]

Total load

Datentyp: loaddata

Das Argument \TLoad beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments \TLoad werden die loaddata in den aktuellen tooldata nicht berücksichtigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.148 MoveLAO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

> **Hinweis**
>
> Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

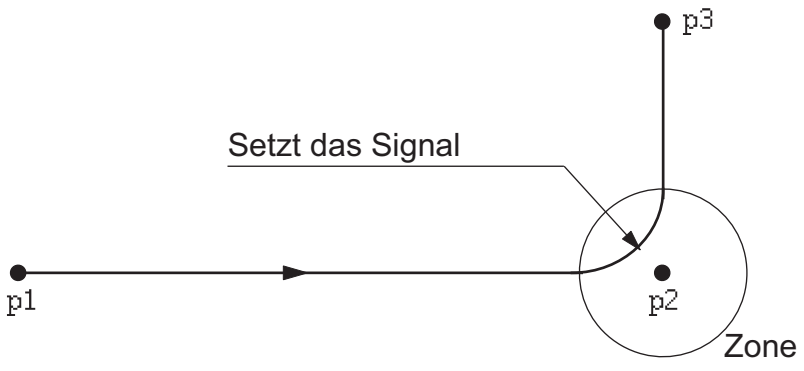
Programmabarbeitung

Weitere Informationen über lineare Bewegungen finden Sie in den Erläuterungen zur Instruktion `MoveL`.

Das analoge Ausgangssignal ist in der Mitte der Zonenbahn für Fly-By-Punkte festgelegt, wie die untere Abbildung zeigt.

Die Abbildung zeigt das Festlegen eines analogen Ausgangssignals in der Zonenbahn mithilfe von `MoveLAO`.

```
MoveLAO p2, v1000, z30, tool2, ao1, 1.1;
```



xx1400001118

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveL` und `SetAO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.148 MoveLAO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

MoveLAO verwendet wird, setzt das analoge Ausgangssignal ein, sobald der Roboter den Stopppunkt erreicht hat.

Das spezifizierte E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt, nicht jedoch, wenn es schrittweise rückwärts bewegt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte Value-Argument für das angegebene analoge Ausgangssignal Signal liegt außerhalb des zulässigen Bereichs.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

MoveLAO kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveLAO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' ] < persistent (PERS) of wobjdata > ','
  [ Signal ':=' ] < variable (VAR) of signalao >] ','
  [ Value ':=' ] < expression (IN) of num > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Lineare Bewegung des Roboters	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.148 MoveLAO - Für eine kreisförmige Bewegung des Roboters und das Festlegen des analogen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Information über	Siehe
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.149 MoveLDO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

Verwendung

`MoveLDO` (*Move Linearly Digital Output*) wird verwendet, um den Werkzeugarbeitspunkt (TCP) linear an ein vorgegebenes Ziel zu bewegen. Das angegebene digitale Ausgangssignal wird in der Mitte der Zonenbahn gesetzt/zurückgesetzt.

Wenn der TCP stationär bleiben soll, kann mit dieser Instruktion auch das Werkzeug umorientiert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveLDO`:

Beispiel 1:

```
MoveLDO p1, v1000, z30, tool2, do1,1;
```

Der TCP von Werkzeug `tool2` wird mit den Geschwindigkeitsdaten `v1000` und den Zonendaten `z30` linear an die Position `p1` bewegt. Ausgang do1 wird in der Mitte der Zonenbahn an `p1` gesetzt.

Argumente

```
MoveLDO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal Value
    [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.149 MoveLDO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`Signal`

Datentyp: `signaldo`

Der Name des digitalen Ausgangssignals, das geändert werden soll.

`Value`

Datentyp: `dionum`

Der gewünschte Signalwert (0 oder 1).

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.149 MoveLDO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

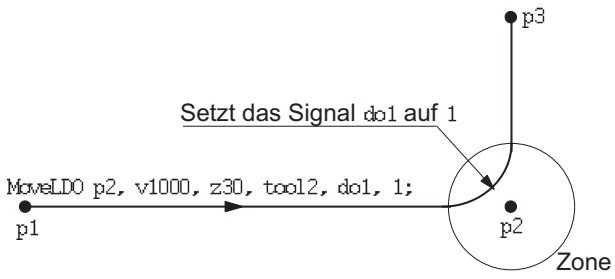
> **Hinweis**
>
> Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Weitere Informationen über lineare Bewegungen finden Sie in den Erläuterungen zur Instruktion `MoveL`.

Das digitale Ausgangsignal wird in der Mitte der Zonenbahn für Fly-By-Punkte gesetzt/zurückgesetzt, wie in der Abbildung unten gezeigt.

Die Abbildung zeigt das Setzen/Zurücksetzen des digitalen Ausgangssignals in der Zonenbahn mithilfe von `MoveLDO`.



xx0500002193

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveL` und `SetDO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveLDO` verwendet wird, wird das digitale Ausgangssignal festgelegt/zurückgesetzt, sobald der Roboter den Stopppunkt erreicht hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.149 MoveLDO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

Das angegebene E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt/zurückgesetzt, aber nicht im Abarbeitungsmodus schrittweise rückwärts.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

MoveLDO kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveLDO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' ] < persistent (PERS) of wobjdata > ','
  [ Signal ':=' ] < variable (VAR) of signaldo >] ','
  [ Value ':=' ] < expression (IN) of dionum > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Lineare Bewegung des Roboters	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.149 MoveLDO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

Information über	Siehe
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.150 MoveLGO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

Verwendung

`MoveLGO`(*Move Linearly Group Output*) wird verwendet um den Werkzeugarbeitspunkt (TCP) linear in Richtung eines vorgegebenen Ziels zu bewegen. Das spezifizierte Gruppenausgangssignal ist in der Mitte der Zonenbahn festgelegt.

Wenn der TCP stationär bleiben soll, kann mit dieser Instruktion auch das Werkzeug umorientiert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MoveLGO`:

Beispiel 1:

```
MoveLGO p1, v1000, z30, tool2, go1 \Value:=5;
```

Der TCP des Werkzeugs `tool2` wird linear in Richtung der Position `p1` mit der Datengeschwindigkeit `v1000` und den Zonendaten `z30` bewegt. Das Gruppenausgangssignal `go1` setzt in der Mitte der Zonenbahn am Zielpunkt `p1` ein.

Argumente

```
MoveLGO ToPoint [\ID] Speed [\T] Zone Tool [\WObj] Signal [\Value]
     | [\DValue] [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.150 MoveLGO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`Signal`

Datentyp: `signalgo`

Der Name des Gruppenausgangs, der geändert werden soll.

`[ \Value ]`

Datentyp: `num`

Der gewünschte Signalwert.

`[ \DValue ]`

Datentyp: `dnum`

Der gewünschte Signalwert.

Wenn keines der Argumente `\Value` oder `\DValue` eingegeben wurde, wird eine Fehlermeldung angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.150 MoveLGO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Weitere Informationen über lineare Bewegungen finden Sie in den Erläuterungen zur Instruktion `MoveL`.

Das analoge Ausgangssignal setzt in der Mitte der Zonenbahn für Fly-By-Punkte ein, wie die untere Abbildung zeigt.

Die Abbildung zeigt das Festlegen eines Gruppenausgangssignals in der Zonenbahn mithilfe von `MoveLGO`.

```
MoveLGO p2, v1000, z30, tool2, go1 \Value:=5;
```

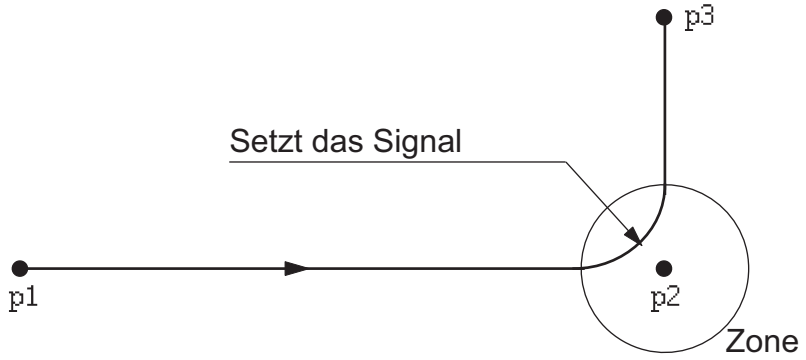
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.150 MoveLGO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung



xx1400001118

Für Stopppunkte wird die normale Programmiersequenz mit `MoveL` und `SetGO` empfohlen. Wenn jedoch der Stopppunkt in der Instruktion `MoveLGO` verwendet wird, setzt der Gruppenausgang ein, sobald der Roboter den Stopppunkt erreicht.

Das spezifizierte E/A-Signal wird im Abarbeitungsmodus kontinuierlich und schrittweise vorwärts gesetzt, nicht jedoch, wenn es schrittweise rückwärts bewegt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

`MoveLGO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveLGO
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' ] < persistent (PERS) of wobjdata > ','
  [ Signal ':=' ] < variable (VAR) of signaldo >] ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.150 MoveLGO - Für eine lineare Bewegung des Roboters und das Festlegen eines digitalen Ausgangs in der Ecke

RobotWare - OS

Fortsetzung

```
[ '\' Value ':=' ] < expression (IN) of num > ]
| [ '\' Dvalue' :=' ] < expression (IN) of dnum >
[ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Lineare Bewegung des Roboters	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen mit E/A-Einstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.151 MoveLSync - Lineares Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur

Verwendung

`MoveLSync` (*Move Linearly Synchronously*) wird verwendet, um den Werkzeugarbeitspunkt (TCP) linear an ein vorgegebenes Ziel zu bewegen. Die angegebene RAPID-Prozedur wird in der Mitte der Zonenbahn am Zielpunkt abgearbeitet.

Wenn der TCP stationär bleiben soll, kann mit dieser Instruktion auch das Werkzeug umorientiert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MoveLSync`:

Beispiel 1:

```
MoveLSync p1, v1000, z30, tool2, "proc1";
```

Der TCP von Werkzeug `tool2` wird mit den Geschwindigkeitsdaten `v1000` und den Zonendaten `z30` linear an die Position `p1` bewegt. Ausgang `proc1` wird in der Mitte der Zonenbahn an `p1` ausgeführt.

Beispiel 2:

```
MoveLSync p1, v1000, z30, tool2, "proc1";
```

Wie in Beispiel 1 oben, jedoch wird hier in der Mitte der Zonenbahn die lokal deklarierte Prozedur `proc1` im Modul `MyModule` aufgerufen.

Argumente

```
MoveLSync ToPoint [\ID] Speed [\T] Zone Tool [\WObj] ProcName
    [\TLoad]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`ProcName`

Procedure Name

Datentyp: `string`

Name der RAPID-Prozedur, die in der Mitte der Zonenbahn am Zielpunkt abgearbeitet werden soll. Der Prozeduraufruf ist ein Aufruf mit später Bindung, daher übernimmt er die Eigenschaften.

Die Prozedur wird auf TRAP-Ebene abgearbeitet (siehe Programmabarbeitung unten).

`[ \TLoad ]`

Total load

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.151 MoveLSync - Lineares Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur
RobotWare - OS
Fortsetzung

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

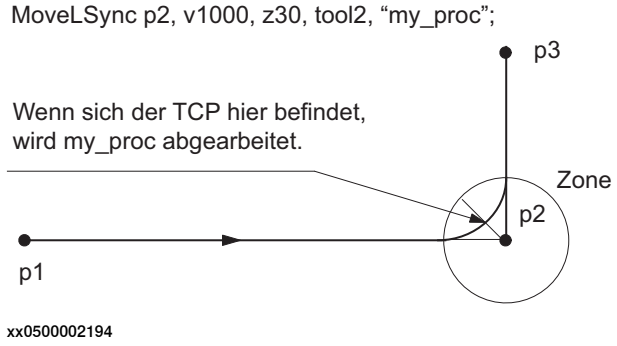
Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Weitere Informationen über lineare Bewegungen finden Sie in den Erläuterungen zur Instruktion `MoveL`.

Die angegebene RAPID-Prozedur wird abgearbeitet, wenn der TCP die Mitte der Zonenbahn am Zielpunkt der Instruktion `MoveLSync` erreicht, wie in der Abbildung unten gezeigt.

Die Abbildung zeigt, dass der Befehl zur Abarbeitung einer anwenderdefinierten RAPID-Prozedur in der Mitte der Zonenbahn erfolgt.



Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Für Stopppunkte wird die Verwendung der normalen Programmiersequenz mit `MoveL` und anderen RAPID-Instruktionen in Folge empfohlen.

Die Tabelle beschreibt die Abarbeitung der angegebenen RAPID-Prozedur in verschiedenen Modi.

Abarbeitungsmodus:	Abarbeitung einer RAPID-Prozedur:
Kontinuierlich oder zyklisch	Gemäß dieser Beschreibung
Vorwärtsschritt	Im Stopppunkt
Rückwärtsschritt	Überhaupt nicht

`MoveLSync` ist eine Einkapselung der Instruktionen `TriggInt` und `TriggL`. Der Prozeduraufruf wird auf TRAP-Ebene abgearbeitet.

Wenn die Mitte der Zonenbahn im Zielpunkt während der Verlangsamung nach einem Programmstopp erreicht wird, wird die Prozedur nicht aufgerufen (Programmabarbeitung wird gestoppt). Der Prozeduraufruf wird beim nächsten Programmstart ausgeführt.

Einschränkungen

Wenn der Roboter die Mitte der Zonenbahn erreicht, wird die RAPID-Routine in der Regel nach einer Verzögerung von 20-30 ms abgearbeitet, je nach Typ der zu diesem Zeitpunkt ausgeführten Bewegung.

Ein Wechsel des Abarbeitungsmodus nach einem Programmstopp von kontinuierlich oder zyklisch zu vorwärts oder rückwärts ergibt einen Fehler. Dieser Fehler teilt dem Anwender mit, dass der Moduswechsel zum Auslassen der RAPID-Prozedur in der Warteschlange für die Abarbeitung auf der Bahn führen kann.

Die Instruktion `MoveLSync` kann nicht auf der Interrupt-Ebene verwendet werden. Die angegebene RAPID-Prozedur kann nicht mit einer schrittweisen Abarbeitung getestet werden.

`MoveLSync` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
MoveLSync
  [ ToPoint ':=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  | [ '\' T ':=' < expression (IN) of num > ] ','
  [ Zone ':=' ] < expression (IN) of zonedata > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj ':=' < persistent (PERS) of wobjdata > ] ','
  [ ProcName ':=' ] < expression (IN) of string > ]
  [ '\' TLoad ':=' < persistent (PERS) of loaddata > ] ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.151 MoveLSync - Lineares Bewegen des Roboters und Abarbeiten einer RAPID-Prozedur

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Lineare Bewegung des Roboters	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Definieren eines positionsabhängigen Interrupts	*TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Lineare Roboterbewegungen mit Ereignissen	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn

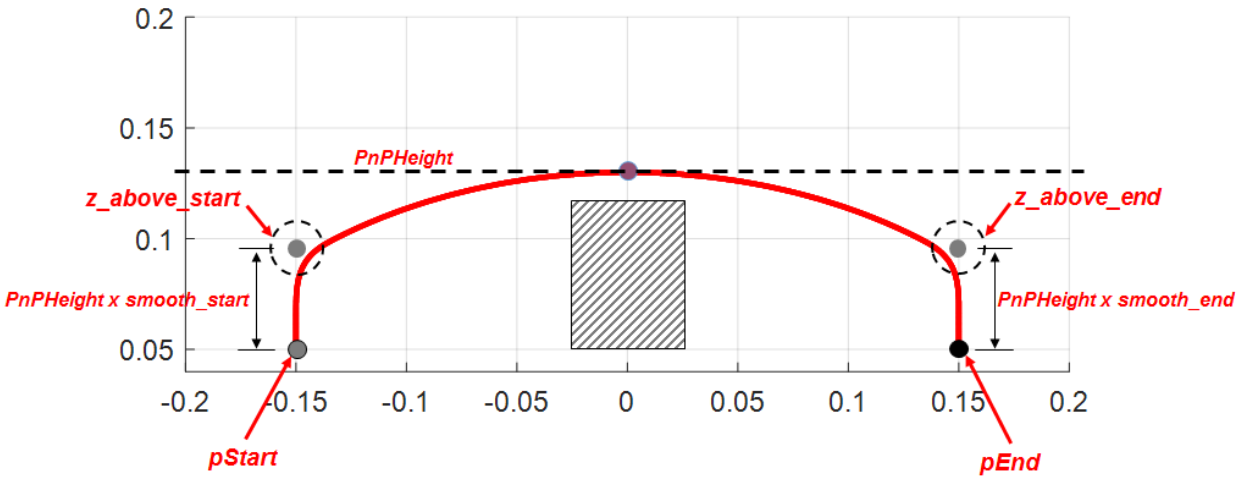
Verwendung

`MovePnP` wird verwendet, um den Werkzeugarbeitspunkt (Tool Center Point, TCP), wie im nachfolgenden Bild angezeigt, schnell entlang einer Aufnahme- und Platzierbahn zu bewegen.

Die Bahn besteht aus zwei vertikalen Bewegungen, die an einem oberen Punkt konvergieren. Die Höhe des oberen Punktes und die Höhe der vertikalen Bewegungen sind konfigurierbar, damit die Bahn verschiedene Formen annehmen kann.

Zu den verschiedenen Formen gehören:

- Form mit fünf Punkten, wie im nachfolgenden Bild angezeigt.
- Form mit vier Punkten, viereckig.
- Form mit drei Punkten, bogenförmig.
- Unsymmetrisch, jede der obigen Kombinationen.



xx1700001194

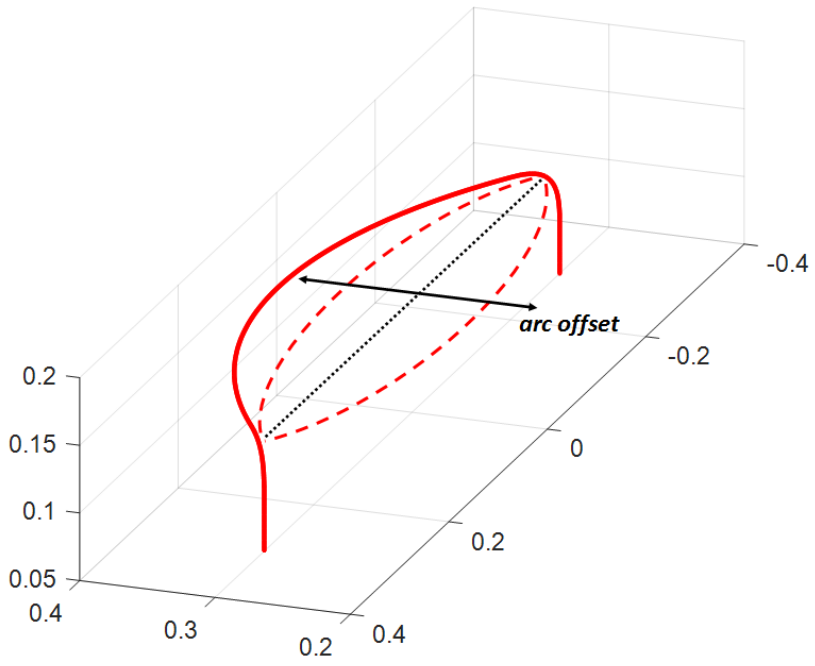
Die Bahn auf der horizontalen Ebene sieht wie ein Bogen aus, dessen Krümmung von einer linearen Bewegung ausgeht durch einen Ausgangsparameter, der nicht

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn
SCARA robots
Fortsetzung

ausdrücklich angegeben werden kann. Dieser Parameter wird intern berechnet, mit dem konkreten Ziel, die Zykluszeit zu minimieren.



xx1800002597

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MovePnP`.

Siehe auch *Weitere Beispiele auf Seite 481*.

Beispiel 1:

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata

my_pnpdata.smooth_start := 50;
my_pnpdata.smooth_end := 50;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata;
```

Fortsetzung auf nächster Seite

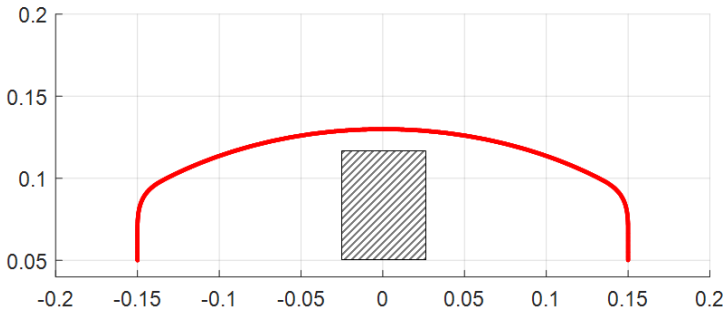
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn

SCARA robots

Fortsetzung

Die Bahn aus einer Form mit fünf Punkten wird konfiguriert, indem die optionalen Parameter `smooth_start` und `smooth_end` auf einen Wert zwischen 0 und 100 eingestellt werden.



xx1700001195

Argumente

```
MovePnP ToPoint [\ID] Speed [\PnPHeight] Zone [\Inpos] Tool [\WObj]
    [\TLoad] [\PnPDataIN] [\SignalIN] [\Value] [\MaxTime]
    [\TimeFlag] [\PnPTrigg] [\PnPTriggOption]
```

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[\ID]`

Synchronization id

Datentyp: `identno`

Das Argument `[\ID]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Die Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeug-Mittelpunktes, der Werkzeugumorientierung und der zusätzlichen Achsen. Der abgearbeitete Pfad mit der `MovePnP`-Instruktion ist für die maximale Geschwindigkeit des Roboters optimiert.

`[\PnPHeight]`

Datentyp: `num`

Die Höhe des höchsten Punkts, angegeben in mm, in absoluten Koordinaten (nicht relativ zum Startpunkt oder zum Endpunkt) im

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn
SCARA robots
Fortsetzung

`ToPoint`-Referenzkoordinatensystem. Wird der Wert nicht angegeben, entspricht die Höhe der oberen Position (Null) der Achse 3.

> **Hinweis**
>
> Die Verwendung von `PnPHeight` ist bei unterschiedlichen Aufnahme- und Platzierhöhen offensichtlich (siehe *Beispiel 4: auf Seite 483*), da `PnPHeight` sowohl von der Anfangs- als auch von der Endhöhe unabhängig sein muss, das heißt, nicht relativ zu beiden festgelegt.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[\Inpos]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an den angegebenen Zielpunkt bewegt wird.

`[\WObj]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn hingegen ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden.

`[\TLoad]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn das \TLoad-Argument auf load0 gesetzt ist, wird das \TLoad-Argument nicht berücksichtigt, und stattdessen werden die loaddata in den aktuellen tooldata verwendet.

Damit das \TLoad-Argument verwendet werden kann, muss der Wert des Systemparameters ModalPayLoadMode unbedingt auf 0 gesetzt werden. Wird ModalPayLoadMode auf 0 eingestellt, kann die GripLoad-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter ModalPayLoadMode auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die loaddata vom Werkzeug in eine bereits vorhandene oder eine neue persistenteloaddata-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang SimMode (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die loaddata im optionalen Argument \TLoad nicht berücksichtigt und stattdessen die loaddata in den aktuellen tooldata verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion GripLoad. Deshalb beträgt der Standardwert für den Systemparameter ModalPayLoadMode 1.

[\PnPDataIN]

Datentyp: pnpdata

pnpdata ist eine Datenstruktur, die zur Konfiguration von Aufnahme- und Platzierbahnen verwendet wird. Bei Nichteingabe wird diese Bahn mit Standardwerten konfiguriert.

Hinweis

Wenn die Struktur deklariert ist, aber nicht alle Felder zugewiesen sind, haben die nicht zugewiesenen Felder standardmäßig Nullwerte.

[\SignalIN]

Datentyp: signaldi

Name des Signals.

[\Value]

Datentyp: dionum

Der gewünschte Wert des Signals.

[\MaxTime]

Datentyp: num

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn

SCARA robots

Fortsetzung

Fehlercode `ERR_WAIT_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

`[\TimeFlag]`

Datentyp: `bool`

Der Ausgabeparameter enthält den Wert `TRUE`, wenn die maximal zulässige Wartezeit abläuft, bevor die Bedingung erfüllt wird. Wenn dieser Parameter in der Instruktion angegeben wird, wird das Ablaufen der `MaxTime` nicht als Fehler betrachtet. Dieses Argument wird ignoriert, wenn das Argument `MaxTime` nicht in der Instruktion angegeben wird.

`[\PnPTrigg]`

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

Hinweis

Die Auslösebedingung wird eingestellt, wenn sich das TCP des Roboters an einem Punkt entlang des Weges gemäß dem Parameter `PnPTriggOption` befindet.

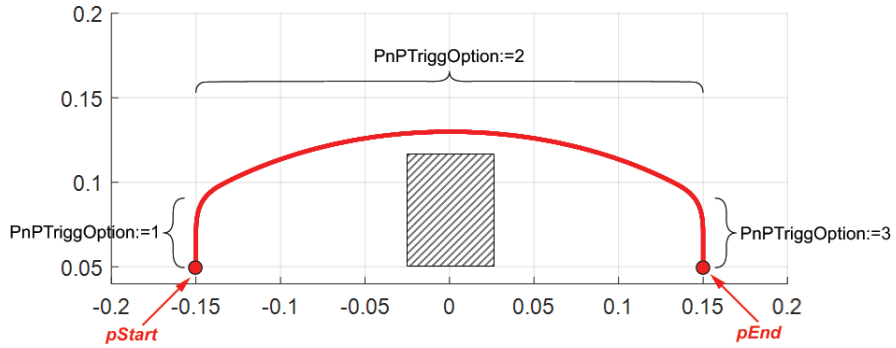
`[\PnPTriggOption]`

Datentyp: `num`

Legt fest, welcher Teil der Bewegung mit den Auslösebedingungen verbunden ist, die durch `PnPTrigg` festgelegt werden.

Die gültigen Optionen sind:

Wert	Beschreibung
1	Die erste vertikale Bewegung, ab dem Startpunkt.
2	Die horizontale Bewegung.
3	Die zweite vertikale Bewegung, in Richtung des Startpunkts.



xx1800001532

Figure 1.1:

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Standardwert ist 2 (z. B. wenn `\PnPTriggOption` weggelassen wird, wird das Auslösen entlang der horizontalen Bewegung festgelegt).

Programmabarbeitung

Der TCP des Roboters wird wie folgt an die Zielposition bewegt:

1. Der TCP bewegt sich vertikal über die aktuelle Position mit einem Abstand, der einem prozentualen Anteil der Bahnhöhe entspricht.
2. Das TCP erreicht den höchsten Punkt, welcher dem Mittelpunkt der Bahn entspricht, mit einer Bahnhöhe, die durch den `\PnPHeight`-Parameter festgelegt ist.
3. Der TCP setzt die Bewegung zu einem Punkt über der Endposition bis zu einer Entfernung fort, die einem prozentualen Anteil der Bahnhöhe entspricht.
4. Der TCP bewegt sich vertikal zur Endposition.

   (Der optionale Parameter `\SignalIN` kann verwendet werden, um auf ein digitales Eingabesignal zu warten, bevor die Bewegung zur Endposition gestartet wird).
5. Das optionale Argument `\PnPTrigg` kann verwendet werden, um die Bewegung mit einem externen Gerät zu synchronisieren, während das TCP entlang des horizontalen Pfads oder entlang der vertikalen Bewegung (je nach dem optionalen Argument `\PnPTriggOption`) verläuft.

Weitere Beispiele

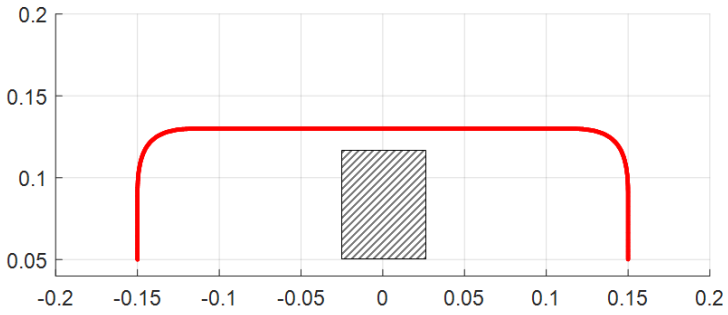
Weitere Beispiele für die Instruktion `MovePnP` sind unten aufgeführt.

Beispiel 1:

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata

my_pnpdata.smooth_start := 100;
my_pnpdata.smooth_end := 100;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata;
```

Die viereckige Form mit vier Punkten wird konfiguriert, indem die optionalen Parameter `smooth_start` und `smooth_end` auf 100 eingestellt werden.



xx1700001196

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn

SCARA robots

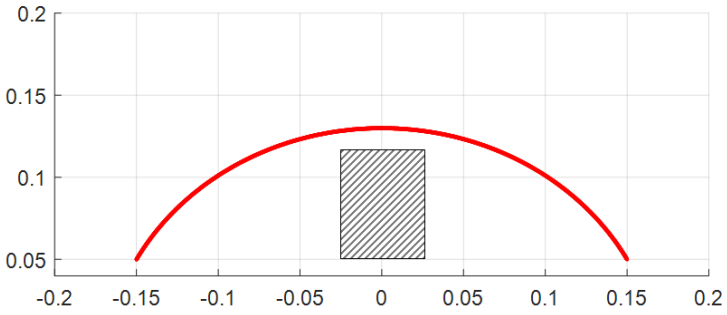
Fortsetzung

Beispiel 2:

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata

my_pnpdata.smooth_start := 0;
my_pnpdata.smooth_end := 0;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata;
```

Form mit drei Punkten, bogenförmig. Wird konfiguriert, indem die optionalen Parameter `smooth_start` und `smooth_end` auf 0 gesetzt werden.



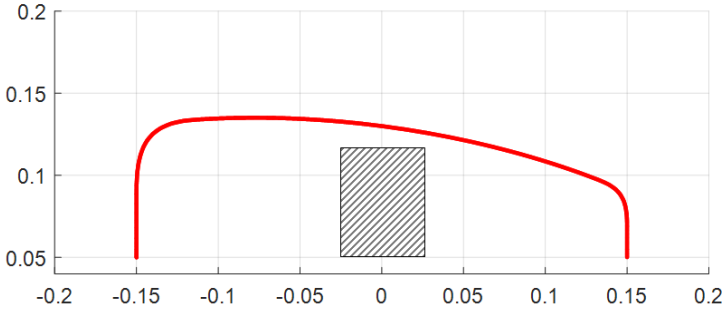
xx1700001197

Beispiel 3

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata

my_pnpdata.smooth_start := 100;
my_pnpdata.smooth_end := 50;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata;
```

Unsymmetrische Form mit vier Punkten. Wird konfiguriert, indem die optionalen Parameter `smooth_start` und `smooth_end` auf verschiedene Werte eingestellt werden.



xx1800000951

Fortsetzung auf nächster Seite

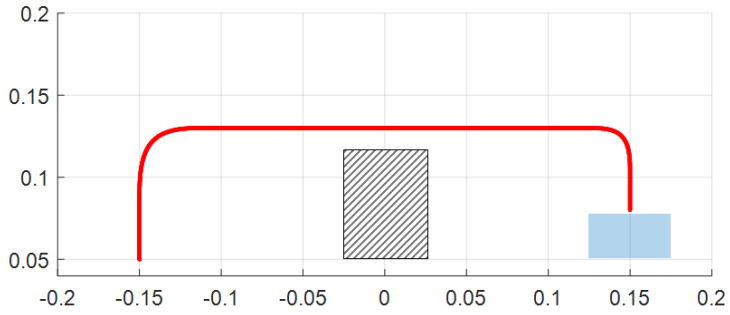
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 4:

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata

my_pnpdata.smooth_start := 100;
my_pnpdata.smooth_end := 100;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata;
```

Form mit vier Punkten und unterschiedlichen Aufnahme- und Platzierebenen. Wird konfiguriert, indem die optionalen Parameter `smooth_start` und `smooth_end` auf 100 gesetzt werden (kein Unterschied von Aufnahme und Platzieren auf derselben Ebene).



xx1800000952

Beispiel 5:

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata

my_pnpdata.smooth_start := 50;
my_pnpdata.smooth_end := 50;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata;
```

Fortsetzung auf nächster Seite

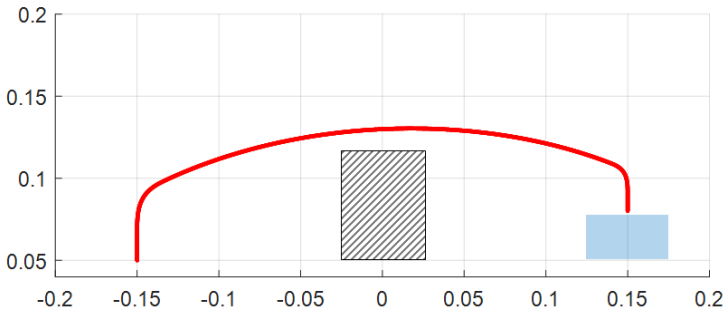
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn

SCARA robots

Fortsetzung

Form mit fünf Punkten und unterschiedlichen Aufnahme- und Platzierebenen. Beachten Sie, dass die Form unsymmetrisch wird, selbst wenn die optionalen Parameter `smooth_start` und `smooth_end` auf denselben Wert gesetzt sind.



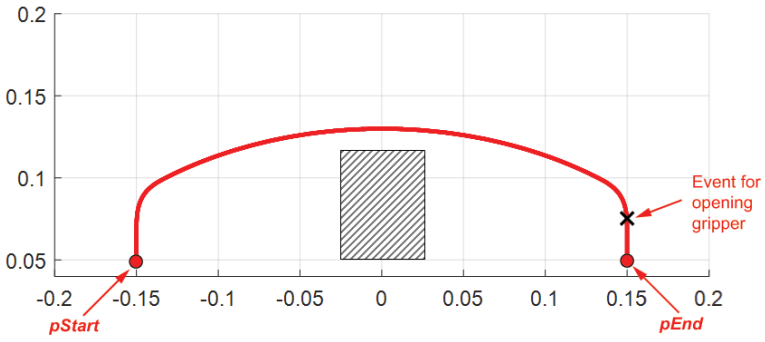
xx1800000953

Beispiel 6:

```
VAR num my_pnp_height := 130;
VAR pnpdata my_pnpdata
VAR triggdata open_gripper;

my_pnpdata.smooth_start := 50;
my_pnpdata.smooth_end := 50;
TriggIO open_gripper, 25 \DOp:=doGripper, 0;
MoveL pStart, v300, fine, tool2;
MovePnP pEnd, v300, \PnPHeight:=my_pnp_height, fine, tool2
     \PnPDataIN:=my_pnpdata \PnPTrigg:=open_gripper
     \PnPTriggOption:=3;
```

Das digitale Ausgangssignal `doGripper` ist auf den Wert `0` gesetzt, wenn sich das TCP in einer Position befindet, in der der vertikale Abstand zu `pEnd` 25 mm beträgt.



xx1800001531

Einschränkungen

Die Instruktion `MovePnP` unterstützt keine Rückwärtsabarbeitung.

Die Instruktion `MovePnP` ist nur für SCARA-Roboter verfügbar.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.152 MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn

SCARA robots

Fortsetzung

Syntax

```
MovePnP
  [ToPoint ':='] <expression (IN) of robtarget>
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' PnPHeight ':=' <expression (IN) of num>]','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]
  ['\' PnPDataIN ':=' <expression (IN) of pnpdata>]
  ['\' SignalIN ':=' <variable (VAR) of signaldi>]
  ['\' Value ':=' <expression (IN) of dionum>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' TimeFlag ':=' <variable (VAR) of bool>]
  ['\' PnPTrigg ':=' <variable (VAR) of triggdata>]
  ['\' PnPTriggOption ':=' <variable (VAR) of num>]';'
```

Weitere Informationen

Information über	Siehe
Konfiguration von Aufnahme- und Platzierbahnen	*pnpdata – Konfiguration von Aufnahme- und Platzierbahnen auf Seite 1755*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.153 MToolRotCalib - Kalibrierung der Rotation eines Werkzeugs in Bewegung

Verwendung

`MToolRotCalib` *(Moving Tool Rotation Calibration)* wird zum Kalibrieren der Drehung eines sich bewegenden Werkzeugs verwendet.

Die Position des Roboters und seine Bewegungen beziehen sich stets auf sein Werkzeug-Koordinatensystem, d. h. den TCP und die Werkzeugorientierung. Für größte Präzision ist es wichtig, das Werkzeug-Koordinatensystem so korrekt wie möglich zu definieren.

Die Kalibrierung kann auch manuell mit dem FlexPendant durchgeführt werden, siehe *Bedienungsanleitung - IRC5 mit FlexPendant*.

Beschreibung

Zur Definition der Werkzeugorientierung benötigen Sie einen festen Punkt im Welt-Koordinatensystem innerhalb des Arbeitsraums des Roboters.

Vor der Verwendung der Instruktion `MToolRotCalib` müssen einige Voraussetzungen erfüllt sein:

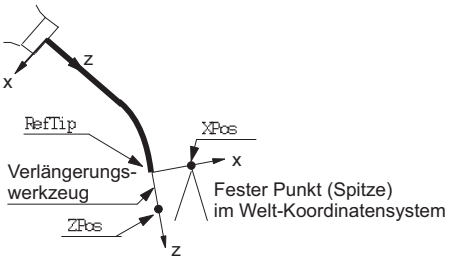
- Das zu kalibrierende Werkzeug muss am Roboter montiert und mit der korrekten Komponente `robhold(TRUE)` definiert sein.
- Wenn der Roboter mit Absolutgenauigkeit verwendet wird, müssen Lasten und Schwerpunkt des Werkzeugs bereits definiert sein. Mit `LoadIdentify` kann die Last definiert werden.
- Der TCP-Wert des Werkzeugs muss bereits definiert sein. Die Kalibrierung kann dann mit der Instruktion `MToolTCPCalib` erfolgen.
- `tool0`, `wobj0`, und `PDispOff` müssen aktiviert sein, bevor der Roboter manuell bewegt wird.
- Bewegen Sie den TCP des aktuellen Werkzeugs so nahe wie möglich an den Ursprungspunkt des Werkzeugkoordinatensystems und definieren Sie ein `jointtarget` für den Referenzpunkt `RefTip`.
- Bewegen Sie den Roboter ohne Ändern der Werkzeugorientierung so, dass der feste Punkt im Welt-Koordinatensystem auf einen Punkt auf der positiven z-Achse des Werkzeug-Koordinatensystems verweist, und definieren Sie ein `jointtarget` für den Punkt `ZPos`.
- Bewegen Sie optional den Roboter ohne Ändern der Werkzeugorientierung so, dass der feste Punkt im Welt-Koordinatensystem auf einen Punkt auf der positiven z-Achse des Werkzeug-Koordinatensystems verweist, und definieren Sie ein `jointtarget` für den Punkt `XPos`.

Als Hilfsmittel zum Anzeigen der positiven z- und x-Achse kann eine Art Verlängerungswerkzeug verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Eine Definition von jointtarget für `RefTip`, `ZPos` und optional `XPos` finden Sie in der folgenden Abbildung.



xx0500002192

Hinweis

Es wird empfohlen, die Positionen `RefTip`, `ZPos` und `XPos` in der Instruktion `MToolRotCalib` nicht zu ändern.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `MToolRotCalib`:

Beispiel 1:

```
! Created with the world fixed tip pointing at origin, positive
! z-axis, and positive x-axis of the wanted tool coordinate
! system.
CONST jointtarget pos_tip := [...];
CONST jointtarget pos_z := [...];
CONST jointtarget pos_x := [...];

PERS tooldata tool1:= [ TRUE, [[20, 30, 100], [1, 0, 0 ,0]], [0.001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0]];

! Instructions for creating or ModPos of pos_tip, pos_z, and pos_x
MoveAbsJ pos_tip, v10, fine, tool0;
MoveAbsJ pos_z, v10, fine, tool0;
MoveAbsJ pos_x, v10, fine, tool0;

! Only tool calibration in the z direction
MToolRotCalib pos_tip, pos_z, tool1;
```

Die Orientatierung des Werkzeugs (`tframe.rot` in z-Richtung von `tool1` wird berechnet. Die x- und y-Richtung der Werkzeugorientierung wird so berechnet, dass sie mit dem Handgelenk-Koordinatensystem übereinstimmt.

Beispiel 2:

```
! Calibration with complete tool orientation
MToolRotCalib pos_tip, pos_z \XPos:=pos_x, tool1;
```

Die vollständige Werkzeugorientierung (`tframe.rot`) von `tool1` wird berechnet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.153 MToolRotCalib - Kalibrierung der Rotation eines Werkzeugs in Bewegung

RobotWare - OS

Fortsetzung

Argumente

```
MToolRotCalib RefTip ZPos [\XPos]Tool
```

`RefTip`

Datentyp: `jointtarget`

Der Punkt, an dem der TCP des Werkzeugs auf den festen Punkt im Welt-Koordinatensystem verweist.

`ZPos`

Datentyp: `jointtarget`

Der Verlängerungspunkt, der die positive z-Richtung definiert.

`[\XPos]`

Datentyp: `jointtarget`

Der Verlängerungspunkt, der die positive x-Richtung definiert. Wenn dieser Punkt ausgelassen wird, sind x- und y-Richtung des Werkzeugs und die entsprechenden Achsen im Handgelenk-Koordinatensystem identisch.

`Tool`

Datentyp: `tooldata`

Die Persistente des Werkzeugs, das kalibriert werden soll.

Programmabarbeitung

Das System berechnet und aktualisiert die Werkzeugorientierung (`tframe.rot`) in den angegebenen Werkzeugdaten. Die Berechnung beruht auf den angegebenen zwei oder drei `jointtarget`-Werten. Die übrigen Daten in Werkzeugdaten wie z. B. TCP (`tframe.trans`) werden nicht geändert.

Syntax

```
MToolRotCalib
  [RefTip ':='] <expression (IN) of jointtarget>','
  [ZPos ':='] <expression (IN) of jointtarget>
  ['\' XPos ':=' <expression (IN) of jointtarget>]','
  [Tool ':='] <persistent (PERS) of tooldata>';'
```

Weitere Informationen

Information über	Siehe
Kalibrierung des TCP für Werkzeuge in Bewegung	*MToolTCPCalib - Kalibrierung des TCP eines Werkzeugs in Bewegung auf Seite 489*
Kalibrierung des TCP für ein stationäres Werkzeug	*SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs auf Seite 829*
Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug	*SToolRotCalib - Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug auf Seite 826*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.154 MToolTCPCalib - Kalibrierung des TCP eines Werkzeugs in Bewegung

Verwendung

`MToolTCPCalib` (*Moving Tool TCP Calibration*) wird zum Kalibrieren des Werkzeugarbeitspunkts (TCP) eines sich bewegenden Werkzeugs verwendet.

Die Position des Roboters und seine Bewegungen beziehen sich stets auf sein Werkzeug-Koordinatensystem, d. h. den TCP und die Werkzeugorientierung. Für größte Präzision ist es wichtig, das Werkzeug-Koordinatensystem so korrekt wie möglich zu definieren.

Die Kalibrierung kann auch manuell mit dem FlexPendant durchgeführt werden, siehe *Bedienungsanleitung - IRC5 mit FlexPendant*.

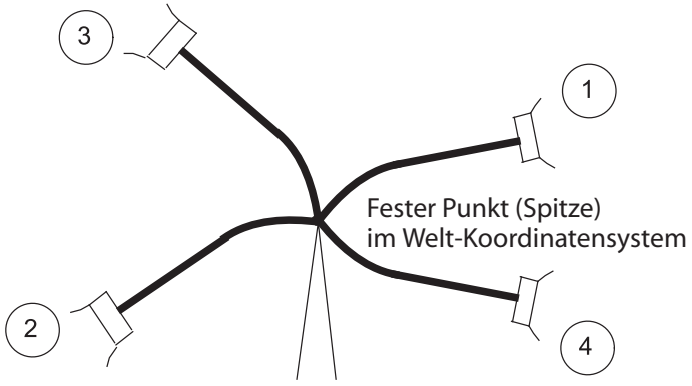
Beschreibung

Um den TCP eines Werkzeugs zu definieren, benötigen Sie einen festen Punkt (Spitze) im Welt-Koordinatensystem innerhalb des Arbeitsraums des Roboters.

Vor der Verwendung der Instruktion `MToolTCPCalib` müssen einige Voraussetzungen erfüllt sein:

- Das zu kalibrierende Werkzeug muss am Roboter montiert und mit der korrekten Komponente `robhold(TRUE)` definiert sein.
- Wenn der Roboter mit Absolutgenauigkeit verwendet wird, müssen Lasten und Schwerpunkt des Werkzeugs bereits definiert sein. Mit `LoadIdentify` kann die Last definiert werden.
- `tool0`, `wobj0`, und `PDispOff` müssen aktiviert sein, bevor der Roboter manuell bewegt wird.
- Bewegen Sie den TCP des aktuellen Werkzeugs so nahe wie möglich an den Ursprungspunkt des Werkzeugkoordinatensystems und definieren Sie ein `jointtarget` für den Referenzpunkt `RefTip`.
- Definieren Sie die restlichen drei Positionen (`p2`,`p3` und `p4`) mit jeweils unterschiedlicher Orientierung.

Definition von vier jointtargets p1 ... p4 (siehe Abbildung unten).



xx0500002191

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.154 MToolTCPCalib - Kalibrierung des TCP eines Werkzeugs in Bewegung

RobotWare - OS

Fortsetzung

> **Hinweis**
>
> Es wird empfohlen, die Positionen `Pos1` bis `Pos4` in der Instruktion `MToolTCPCalib` nicht zu ändern.
>
> Die Umorientierung zwischen den 4 Positionen sollte so groß wie möglich sein, so dass der Roboter unterschiedliche Konfigurationen erhält. Es empfiehlt sich außerdem, nach einer Kalibrierung die Qualität des TCP zu überprüfen. Dies kann durch Umorientierung des Werkzeugs erfolgen, um zu überprüfen, ob der TCP nicht in Bewegung ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `MToolTCPCalib`:

Beispiel 1:

```
! Created with actual TCP pointing at the world fixed tip
CONST jointtarget p1 := [...];
CONST jointtarget p2 := [...];
CONST jointtarget p3 := [...];
CONST jointtarget p4 := [...];

PERS tooldata tool1:= [TRUE, [[0, 0, 0], [1, 0, 0 ,0]], [0.001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0]];
VAR num max_err;
VAR num mean_err;
...
! Instructions for createing or ModPos of p1 - p4
MoveAbsJ p1, v10, fine, tool0;
MoveAbsJ p2, v10, fine, tool0;
MoveAbsJ p3, v10, fine, tool0;
MoveAbsJ p4, v10, fine, tool0;
...
MToolTCPCalib p1, p2, p3, p4, tool1, max_err, mean_err;
```

Der TCP-Wert (`tframe.trans`) von `tool1` wird kalibriert und aktualisiert. `max_err` und `mean_err` geben die maximale Abweichung in mm bzw. die mittlere Abweichung in mm vom errechneten TCP an.

Argumente

```
MToolTCPCalib Pos1 Pos2 Pos3 Pos4 Tool MaxErr MeanErr
```

`Pos1`

Datentyp: `jointtarget`

Der erste Näherungspunkt.

`Pos2`

Datentyp: `jointtarget`

Der zweite Näherungspunkt.

`Pos3`

Datentyp: `jointtarget`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der dritte Näherungspunkt.

`Pos4`

Datentyp: `jointtarget`

Der vierte Näherungspunkt.

`Tool`

Datentyp: `tooldata`

Die Persistente des Werkzeugs, das kalibriert werden soll.

`MaxErr`

Datentyp: `num`

Der maximale Fehler in mm für einen Näherungspunkt.

`MeanErr`

Datentyp: `num`

Der durchschnittliche Abstand der Näherungspunkte vom berechneten TCP, d. h., wie präzise der Roboter in Relation zur Spitze positioniert war.

Programmabarbeitung

Das System berechnet und aktualisiert den TCP-Wert im Handgelenk-Koordinatensystem (`tframe.trans`) in den angegebenen `tooldata`. Die Berechnung beruht auf den angegebenen vier `jointtarget`-Werten. Die übrigen Daten in tooldata wie z. B. die Werkzeugorientierung (`tframe.rot`) werden nicht geändert.

Syntax

```
MToolTCPCalib
  [Pos1 ':='] <expression (IN) of jointtarget>','
  [Pos2 ':='] <expression (IN) of jointtarget>','
  [Pos3 ':='] <expression (IN) of jointtarget>','
  [Pos4 ':='] <expression (IN) of jointtarget>','
  [Tool ':='] <persistent (PERS) of tooldata>','
  [MaxErr ':='] <variable (VAR) of num>','
  [MeanErr ':='] <variable (VAR) of num>';'
```

Weitere Informationen

Information über	Siehe
Kalibrierung der Rotation für Werkzeuge in Bewegung	*MToolRotCalib - Kalibrierung der Rotation eines Werkzeugs in Bewegung auf Seite 486*
Kalibrierung des TCP für ein stationäres Werkzeug	*SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs auf Seite 829*
Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug	*SToolRotCalib - Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug auf Seite 826*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.155 Open - Öffnet eine Datei oder ein E/A-Gerät

Verwendung

`Open` wird verwendet, um eine Datei oder einen seriellen Kanal zum Lesen oder Schreiben zu öffnen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Open`:

Siehe auch *Weitere Beispiele auf Seite 494*.

Beispiel 1:

```
VAR iodev logfile;
...
Open "HOME:" \File:= "LOGFILE1.DOC", logfile \Write;
```

Die Datei `LOGFILE1.DOC` in `HOME:` wird zum Schreiben geöffnet. Der Verweisname `logfile` wird später im Programm beim Schreiben in die Datei verwendet.

Beispiel 2:

```
VAR iodev logfile;
...
Open "LOGFILE1.DOC", logfile \Write;
```

Dasselbe Ergebnis wie in Beispiel 1. Das Standardverzeichnis ist `HOME:`.

Argumente

```
Open Object [\File] IODevice [\Read] | [\Write] | [\Append] [\Bin]
```

`Object`

Datentyp: `string`

Das E/A-Objekt (E/A-Einheit), das geöffnet werden soll, z. B. `"HOME:"`, `"TEMP:"`, `"com1:"` oder `"pc:"` (Option).

Die Tabelle beschreibt andere E/A-Geräte an der Robotersteuerung.

Name der E/A-Einheit	Typ der E/A-Einheit
`"HOME:"` oder `diskhome` [i]	SD-Speicherkarte
`"TEMP:"` oder `disktemp` [i]	SD-Speicherkarte
`"RemovableDisk1:"` oder `usbdisk1` [i] `"RemovableDisk2:"` oder `usbdisk2` [i] `"RemovableDisk3:"` oder `usbdisk3` [i]	z.B. USB-Memorystick
`"com1:"` [ii]	Serieller Kanal
`"pc:"` [iii]	Bereitgestellter Datenträger
`"RAMDISK:"` oder `diskram` [i], [iv]	RAM-Disk-Arbeitsspeicher

i RAPID-Zeichenfolge, die einen Gerätenamen definiert.
ii Benutzerdefinierter Name des seriellen Kanals, in den Systemparametern definiert.
iii Applikationsprotokoll, Serverpfad, in den Systemparametern definiert.
iv Der RAM-Disk-Arbeitsspeicher ist nicht zur dauerhaften Speicherung von Daten geeignet. Er hat ungefähr 100 MB Speicherplatz und wird beim Herunterfahren jedes Mal gelöscht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die folgende Tabelle beschreibt andere E/A-Geräte an der virtuellen Steuerung.

Name der E/A-Einheit	Typ der E/A-Einheit
"HOME:" oder diskhome [i]	
"TEMP:" oder disktemp	Festplatte
"RemovableDisk1:" oder usbdisk1 "RemovableDisk2:" oder usbdisk2 "RemovableDisk3:" oder usbdisk3	z.B. USB-Memorystick
"RAMDISK:" oder diskram[i]	Festplatte ..\TEMP (verweist auf den TEMP-Ordner, der sich im selben Ordner wie das virtuelle System befindet)

[i] RAPID-Zeichenfolge, die einen Gerätenamen definiert.

[\File]

Datentyp: string

Der Name der zu öffnenden Datei, z. B. "LOGFILE1.DOC" oder "LOGDIR/LOGFILE1.DOC"

Der vollständige Pfad kann auch im Argument Object, "HOME:/LOGDIR/LOGFILE.DOC". angegeben werden.

IODevice

Datentyp: iodev

Ein Verweis auf die Datei oder das E/A-Gerät, die/das geöffnet werden soll. Dieser Verweis wird verwendet, um eine Datei oder ein E/A-Gerät zum Lesen oder Schreiben zu öffnen.

[\Read]

Datentyp: switch

Öffnet eine Datei oder ein E/A-Gerät zum Lesen. Beim Lesen aus einer Datei erfolgt der Vorgang ab dem Beginn der Datei.

[\Write]

Datentyp: switch

Öffnet eine Datei oder ein E/A-Gerät zum Schreiben. Wenn die gewählte Datei bereits vorhanden ist, wird ihr Inhalt gelöscht. Das anschließende Schreiben erfolgt ab dem Beginn der Datei.

[\Append]

Datentyp: switch

Öffnet eine Datei oder ein E/A-Gerät zum Schreiben. Wenn die gewählte Datei bereits vorhanden ist, erfolgt anschließendes Schreiben am Ende der Datei.

Öffnen Sie eine Datei mit dem \Append-Argument und ohne das \Bin-Argument. Die Instruktion öffnet eine zeichenorientierte Datei oder ein E/A-Gerät zum Schreiben.

Eine Datei oder ein E/A-Gerät mit \Append.und \Bin-Argumenten öffnen. Die Instruktion öffnen eine Binärdatei oder ein E/A-Gerät für Lesen und Schreiben. Die

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.155 Open - Öffnet eine Datei oder ein E/A-Gerät
RobotWare - OS
Fortsetzung

Argumente `\Read`, `\Write`, `\Append` schließen sich gegenseitig aus. Wenn keines dieser Argumente angegeben ist, verhält sich die Instruktion wie mit dem Argument `\Write` für zeichenorientierte Dateien oder ein E/A-Gerät (Instruktion ohne Argument `\Bin`) und wie mit dem Argument \Append für Binärdateien oder ein E/A-Gerät (Instruktion mit Argument `\Bin`).

`[\Bin]`

Datentyp: `switch`

Die Datei bzw. das E/A-Gerät wird in binärem Modus geöffnet. Wenn keines der Argumente `\Read`, `\Write` oder `\Append` angegeben ist, öffnet die Instruktion eine Binärdatei oder einen seriellen Kanal zum Lesen und Schreiben mit dem Dateizeiger am Ende der Datei.

Mit der `Rewind`-Instruktion kann der Dateizeiger bei Bedarf auf den Anfang der Datei gesetzt werden.

Die Instruktionen für den Zugriff auf eine binäre Datei oder ein E/A-Gerät unterscheiden sich von den Instruktionen für den Zugriff auf eine zeichenorientierte Datei.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `Open` sind unten aufgeführt.

Beispiel 1:

```
]
VAR iodev printer;
...
Open "com1:", printer \Bin;
WriteStrBin printer, "This is a message to the printer\0D";
Close printer;
```

Der serielle Kanal `com1:` wird zum binären Lesen und Schreiben geöffnet. Der Referenzname `printer` wird später beim Schreiben auf den seriellen Kanal und bei dessen Schließen verwendet.

Beispiel 2

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num float := 0.2;
VAR string answer;

ClearRawBytes raw_data_out;
PackDNHeader "10", "20 1D 24 01 30 64", raw_data_out;
PackRawBytes float, raw_data_out, (RawBytesLen(raw_data_out)+1)
     \Float4;

Open "/FCI1:/dsqc328_1", io_device \Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in \Time:=1;
Close io_device;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
UnpackRawBytes raw_data_in, 1, answer \ISOLatin1Encoding:=10;
```

In diesem Beispiel wird `raw_data_out` gelöscht und dann mit einem DeviceNet-Header und dem Float-Wert 0.2 gepackt.

Ein Gerät `"/FCI1/:dsqc328_1"` wird geöffnet und die aktuellen gültigen Daten in `raw_data_out` werden auf das Gerät geschrieben. Dann wartet das Programm höchstens 1 Sekunde auf das Lesen von dem Gerät, das in `raw_data_in` gespeichert ist.

Nachdem das Gerät „`/FCI1/:dsqc328_1`“ geschlossen wurde, werden die gelesenen Daten als Zeichenfolge von `10`-Zeichen entpackt und in der Zeichenfolge `answer` gespeichert.

Programmabarbeitung

Die angegebene Datei oder das E/A-Gerät wird geöffnet, um das Lesen und Schreiben darauf zu ermöglichen.

Dieselbe physische Datei kann mehrmals gleichzeitig geöffnet werden, jeder Aufruf der Instruktion `Open` liefert jedoch einen unterschiedlichen Verweis auf die Datei (Datentyp `iodev`). Beispielsweise kann gleichzeitig ein Zeiger zum Lesen und ein anderer Zeiger zum Schreiben in der Datei verwendet werden.

Die beim Öffnen einer Datei oder eines E/A-Geräts verwendete Variable „iodev“ muss frei verfügbar sein. Wenn die Variable zuvor für das Öffnen einer Datei verwendet wurde, muss diese Datei geschlossen werden, bevor eine neue `Open`-Instruktion mit derselben iodev-Variable möglich ist.

Bei einem Programmstopp und beim Verschieben des PZ auf Main wird jede geöffnete Datei und jedes geöffnete E/A-Gerät in der Programmtask geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt. Ein Ausnahme von der Regel sind Variablen, die als gemeinsam genutzte Variablen vom Typ globale VAR oder LOCAL VAR im System installiert sind. Ein solche Datei oder ein solches E/A-Gerät, die bzw. das für das gesamte System gültig ist, bleibt geöffnet.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEOPEN`	Eine Datei kann nicht geöffnet werden.

Syntax

```
Open
  [ Object ':='] <expression (IN) of string>
  [ '\' File ':=' <expression (IN) of string>] ','
  [ IODevice ':='] <variable (VAR) of iodev>
  [ '\' Read] |
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.155 Open - Öffnet eine Datei oder ein E/A-Gerät

RobotWare - OS

Fortsetzung

```
[ '\' Write] |
[ '\' Append]
[ '\' Bin] ';'
```

Weitere Informationen

Information über	Siehe
Schreiben in, Lesen aus und Schließen von Dateien oder E/A-Geräten	*Technisches Referenzhandbuch - RAPID Overview*
Fieldbus Command Interface Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.156 OpenDir - Öffnen eines Verzeichnisses

Verwendung

`OpenDir` wird verwendet, um die Inhalte eines Verzeichnisses anzuzeigen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `OpenDir`:

Beispiel 1:

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
  CloseDir directory;
ENDPROC
```

Dieses Beispiel gibt die Namen aller Dateien bzw. Unterverzeichnisse unter dem angegebenen Verzeichnis aus.

Argumente

```
OpenDir Dev Path
```

`Dev`

Datentyp: `dir`

Eine Variable mit Verweis auf das Verzeichnis, die mithilfe der Instruktion `OpenDir` abgerufen wurde. Diese Variable wird dann verwendet, um die Inhalte des Verzeichnisses auszulesen.

`Path`

Datentyp: `string`

Der Pfad des Verzeichnisses.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Der Pfad zeigt auf ein nicht vorhandenes Verzeichnis oder es sind zu viele Verzeichnisse gleichzeitig geöffnet.

Einschränkungen

Geöffnete Verzeichnisse sollten vom Benutzer nach dem Lesen immer geschlossen werden (Instruktion `CloseDir`).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.156 OpenDir - Öffnen eines Verzeichnisses

RobotWare - OS

Fortsetzung

Syntax

```
OpenDir
  [ Dev ':=' ] < variable (VAR) of dir> ','
  [ Path ':=' ] < expression (IN) of string> ';'
```

Weitere Informationen

Information über	Siehe
Verzeichnis	*dir – Dateiverzeichnisstruktur auf Seite 1689*
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Lesen eines Verzeichnisses	*ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis auf Seite 1459*
Schließen eines Verzeichnisses	*CloseDir - Schließen eines Verzeichnisses auf Seite 153*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.157 PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten

Verwendung

`PackDNHeader` wird verwendet, um den Header einer expliziten DeviceNet-Meldung in einen Container vom Typ `rawbytes` zu packen.

Der Datenteil der DeviceNet-Meldung kann anschließend mithilfe der Instruktion `PackRawBytes` festgelegt werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `PackDNHeader`:

Beispiel 1:

```
VAR rawbytes raw_data;

PackDNHeader "0E", "6,20 01 24 01 30 06,9,4", raw_data;
```

Packt den Header für die explizite DeviceNet-Meldung mit dem Servicecode `"0E"` und der Bahnzeichenfolge `"6,2001 24 01 30 06,9,4"` in die entsprechenden `raw_data`, um die Seriennummer von einem E/A-Gerät abzurufen.

Diese Meldung ist bereit zum Senden, ohne zusätzliche Daten in die Meldung eintragen zu müssen.

Beispiel 2:

```
VAR rawbytes raw_data;

PackDNHeader "10", "20 1D 24 01 30 64", raw_data;
```

Packt den Header für die explizite DeviceNet-Meldung mit dem Servicecode `"10"` und der Bahnzeichenfolge `"201D 24 01 30 64"` in die entsprechenden `raw_data`, um die Filterzeit für die ansteigende Kante an Eingangssignal 1 für ein E/A-Gerät einzustellen.

Dieser Meldung müssen Daten für die Filterzeit hinzugefügt werden. Dies ist mithilfe der Instruktion `PackRawBytes` beginnend ab Index `RawBytesLen(raw_data)+1` (im Anschluss an `PackDNHeader`) möglich.

Argumente

```
PackDNHeader Service Path RawData
```

`Service`

Datentyp: `string`

Der auszuführende Service (z. B. Abrufen oder Einstellen des Attributs). Muss als Zeichenfolge im Hexadezimalformat (z. B. `"IF"`) angegeben werden.

Zeichenfolgenlänge:	2 Zeichen
Format	'0' -' 9', 'a' -'f', 'A' - 'F'
Bereich	"00" - "FF"

Die Werte für `Service` befinden sich in der EDS-Datei. Weitere Beschreibungen finden Sie bei der Open DeviceNet Vendor Association *ODVA DeviceNet Specification revision 2.0*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.157 PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten

RobotWare - OS

Fortsetzung

`Path`

Datentyp: `string`

Die Werte für `Path` befinden sich in der EDS-Datei. Weitere Beschreibungen finden Sie bei der Open DeviceNet Vendor Association *ODVA DeviceNet Specification, Revision 2.0*.

Unterstützt werden sowohl lange Zeichenfolgenformate (z. B. "`6,20 1D 24 01 30 64,8,1`") als auch kurze (z. B. "`20 1D 24 01 30 64`").

`RawData`

Datentyp: `rawbytes`

Der Variablencontainer, in den die Daten des Header der Meldung ab Index 1 in `RawData` gepackt werden sollen.

Programmabarbeitung

Während der Programmabarbeitung wird der `RawData`-Container der DeviceNet-Meldung:

- zunächst vollständig gelöscht
- und anschließend wird der Header-Teil mit Daten gepackt.

Format des DeviceNet-Header

Die Instruktion `PackDNHeader` erstellt einen Header einer DeviceNet-Meldung in folgendem Format:

Format des RawData-Headers	Byteanzahl	Hinweis
Format	1	Interner Robotersteuerungscode für DeviceNet
Service	1	Hex-Code für Service
Länge der Bahn	1	In Byte
Bahn	x	ASCII-Zeichen

Der Datenteil der DeviceNet-Meldung kann anschließend mithilfe der Instruktion `PackRawBytes` beginnend ab dem Index mithilfe von `(RawBytesLen(my_rawdata)+1)` abgerufen werden.

Syntax

```
PackDNHeader
  [Service ':='] <expression (IN) of string>','
  [Path ':='] <expression (IN) of string>','
  [RawData ':='] <variable (VAR) of rawbytes>';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Bit/Byte-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.158 PackRawBytes - Packen von Daten in rawbytes-Daten

Verwendung

`PackRawBytes` wird verwendet, um den Inhalt von Variablen des Typs `num`, `dnum`, `byte` oder `string` in einen Container vom Typ `rawbytes` zu packen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `PackRawBytes`.

```
VAR rawbytes raw_data;
VAR num integer := 8;
VAR dnum bigInt := 4294967295;
VAR num float := 13.4;
VAR byte data1 := 122;
VAR byte byte1;
VAR string string1:="abcdefg";
PackDNHeader "10", "20 1D 24 01 30 64", raw_data;
```

Packen Sie den Header für DeviceNet in `raw_data`.

Packen Sie anschließend die geforderten Feldbusdaten in `raw_data` mithilfe von `PackRawBytes`. Das folgende Beispiel zeigt, wie unterschiedliche Daten hinzugefügt werden können.

Beispiel 1:

```
PackRawBytes integer, raw_data, (RawBytesLen(raw_data)+1) \IntX :=
     DINT;
```

Der Inhalt der nächsten 4 Byte nach dem Header in `raw_data` ist `8` dezimal.

Beispiel 2:

```
PackRawBytes bigInt, raw_data, (RawBytesLen(raw_data)+1) \IntX :=
     UDINT;
```

Der Inhalt der nächsten 4 Byte nach dem Header in `raw_data` ist `4294967295` dezimal.

Beispiel 3

```
PackRawBytes bigInt, raw_data, (RawBytesLen(raw_data)+1) \IntX :=
     LINT;
```

Der Inhalt der nächsten 8 Byte nach dem Header in `raw_data` ist `4294967295` dezimal.

Beispiel 4:

```
PackRawBytes float, raw_data, RawBytesLen(raw_data)+1) \Float4;
```

Der Inhalt der nächsten 4 Byte in `raw_data` ist 13,4 dezimal.

Beispiel 5:

```
PackRawBytes data1, raw_data, (RawBytesLen(raw_data)+1) \ASCII;
```

Der Inhalt des nächsten Byte in `raw_data` ist `122`, der ASCII-Code für "z".

Beispiel 6:

```
PackRawBytes string1, raw_data, (RawBytesLen(raw_data)+1) \ASCII;
```

Der Inhalt der nächsten 7 Byte in `raw_data` ist `"abcdefg"` (in ASCII-Code).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 7:

```
byte1 := StrToByte("1F" \Hex);
PackRawBytes byte1, raw_data, (RawBytesLen(raw_data)+1) \Hex1;
```

Der Inhalt des nächsten Byte in `raw_data` ist `"1F"` (hexadezimal).

Argumente

```
PackRawBytes Value RawData [ \Network ] StartIndex [\Hex1] | [\IntX
     ] | [ \Float4 ] | [ \ASCII ]
```

Siehe *Argumente kombinieren auf Seite 504*.

`Value`

Datentyp: `anytype`

In `RawData` zu packende Daten.

Mögliche Datentypen: `num`, `dnum`, `byte` oder `string`. Datenfelder können nicht verwendet werden.

`RawData`

Datentyp: `rawbytes`

Der Variablencontainer, in den die Daten gepackt werden sollen.

`[ \Network ]`

Datentyp: `switch`

Zeigt an, dass die Werte von `integer` und `float` in `RawData` in "Big endian" (Netzwerkreihenfolge) gepackt werden sollen. Big endian wird in ProfiBus und InterBus verwendet.

Ohne diesen Schalter werden Daten vom Typ `integer` und `float` in Little-endian-Darstellung (keine Netzwerkreihenfolge) in `RawData` gepackt. Little-endian wird in DeviceNet verwendet.

Nur relevant gemeinsam mit dem Optionsparameter `\IntX` - `UINT`, `UDINT`, `ULINT`, `INT`, `DINT`, `LINT` und `\Float4`.

`StartIndex`

Datentyp: `num`

`StartIndex` (Wert zwischen 1 und 1024) gibt an, wo das erste Byte aus `Value` in `RawData` angeordnet werden soll.

`[ \Hex1 ]`

Datentyp: `switch`

Der zu packende `Value` liegt im Format `byte` vor und soll ins Hexadezimalformat konvertiert und in 1 Byte in `RawData` gespeichert werden.

`[ \IntX ]`

Datentyp: `inttypes`

Der zu packende `Value` hat das Format `num` oder `dnum`. Es handelt sich um eine Ganzzahl, und diese wird entsprechend der angegebenen Konstante vom Datentyp `inttypes` in `RawData` gespeichert.

Siehe *Vordefinierte Daten auf Seite 505*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.158 PackRawBytes - Packen von Daten in rawbytes-Daten

RobotWare - OS

Fortsetzung

`[ \Float4 ]`

Datentyp: `switch`

Der zu packende `Value` liegt im Format `num` vor und soll als Fließkommazahl in 4 Byte in `RawData` gespeichert werden.

`[ \ASCII ]`

Datentyp: `switch`

Der zu packende `Value` liegt im Format `byte` oder `string` vor.

Wenn der zu packende `Value` im Format `byte` vorliegt, wird er in `RawData` in 1 Byte gespeichert, wobei der `Value` als ASCII-Code für ein Zeichen interpretiert wird.

Wenn der zu packende `Value` im Format „string“ vorliegt (1 bis 80 Zeichen), wird er in `RawData` als ASCII-Zeichen mit derselben Anzahl an Zeichen wie in `Value` gespeichert. String-Daten werden in Daten vom Typ `rawbytes` nicht durch das System `NULL`-terminiert. Es ist Aufgabe des Programmierers, falls nötig einen String-Header hinzuzufügen (erforderlich für DeviceNet).

Argumente kombinieren

Eines der Argumente `\Hex1`, `\IntX`, `\Float4` oder `\ASCII` muss verwendet werden.

Die folgenden Kombinationen sind möglich:

Datentyp `Value`:	Mögliche optionale Parameter:
`num` [i]	`\IntX`
`dnum` [ii]	`\IntX`
`num`	`\Float4`
`string`	`\ASCII` (1 bis 80 Zeichen)
`byte`	`\Hex1 \ASCII`

[i] Muss eine Ganzzahl sein innerhalb des Wertebereichs der ausgewählten symbolischen Konstante `USINT`, `UINT`, `UDINT`, `SINT`, `INT` oder `DINT`.

[ii] Muss eine Ganzzahl sein innerhalb des Wertebereichs der ausgewählten symbolischen Konstante `USINT`, `UINT`, `UDINT`, `ULINT`, `SINT`, `INT`, `DINT` oder `LINT`.

Programmabarbeitung

Bei der Programmabarbeitung werden Daten aus der Variablen vom Typ `anytype` in einen Container vom Typ `rawbytes` gepackt.

Die aktuelle Länge gültiger Bytes in der Variablen `RawData` wird gesetzt auf:

- (`StartIndex` + packed_number_of_bytes - 1)
- Die aktuelle Länge gültiger Bytes in der Variablen `RawData` wird nicht geändert, wenn der vollständige Packvorgang innerhalb der alten aktuellen Länge gültiger Bytes in der Variablen `RawData` durchgeführt wird,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `inttypes` sind vordefiniert und können zur Angabe der Ganzzahl im Parameter `\IntX` verwendet werden.

Symbolische Konstante	Konstanter Wert	Ganzzahlformat	Wertebereich der Ganzzahl
`USINT`	1	Vorzeichenlose 1-Byte-Ganzzahl	0 ... 255
`UINT`	2	Vorzeichenlose 2-Byte-Ganzzahl	0 ... 65 535
`UDINT`	4	Vorzeichenlose 4-Byte-Ganzzahl	0 ... 8 388 608 [i] 0 ... 4 294 967 295 [ii]
`ULINT`	8	Vorzeichenlose 8-Byte-Ganzzahl	0 ... 4 503 599 627 370 496 [iii]
`SINT`	- 1	Vorzeichenbehaftete 1-Byte-Ganzzahl	- 128... 127
`INT`	- 2	Vorzeichenbehaftete 2-Byte-Ganzzahl	- 32 768 ... 32 767
`DINT`	- 4	Vorzeichenbehaftete 4-Byte-Ganzzahl	- 8 388 607 ... 8 388 608 [i] -2 147 483 648 ... 2 147 483 647 [iv]
`LINT`	- 8	Vorzeichenbehaftete 8-Byte-Ganzzahl	- 4 503 599 627 370 496... 4 503 599 627 370 496 [iii]

i Begrenzung in RAPID für die Speicherung einer Ganzzahl vom Datentyp `num`.
ii Bereich bei Verwendung einer dnum-Variablen und inttype `UDINT`.
iii Begrenzung in RAPID für die Speicherung einer Ganzzahl vom Datentyp `dnum`.
iv Bereich bei Verwendung einer dnum-Variablen und inttype `DINT`.

Syntax

```
PackRawBytes
  [Value ':=' ] <expression (IN) of anytype>','
  [RawData ':='] <variable (VAR) of rawbytes>
  ['\' Network] ','
  [StartIndex ':='] <expression (IN) of num>
  ['\' Hex1]
  |['\' IntX ':=' <expression (IN) of inttypes>]
  |['\' Float4]
  |['\' ASCII] ';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.158 PackRawBytes - Packen von Daten in rawbytes-Daten

RobotWare - OS

Fortsetzung

Information über	Siehe
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Bit/Byte-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.159 PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn

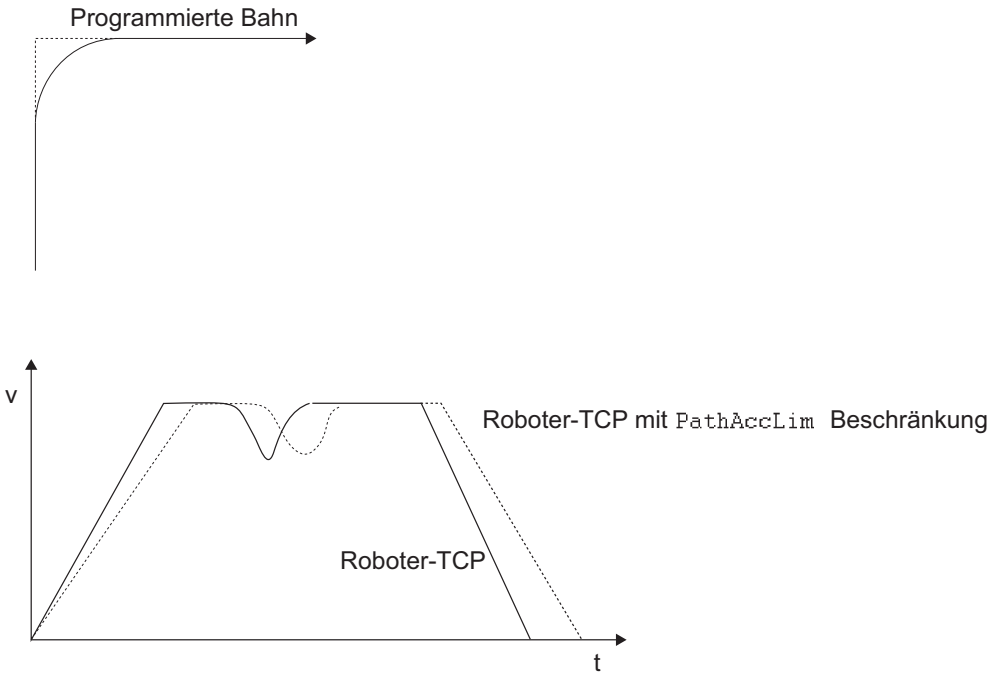
Verwendung

`PathAccLim` (*Path Acceleration Limitation*) wird verwendet, um die Beschränkungen für die TCP-Beschleunigung und/oder TCP-Verzögerung entlang der Bewegungsbahn zu setzen oder zurückzusetzen.

Die Beschränkung erfolgt entlang der Bewegungsbahn, d. h. der Beschleunigung im Bahn-Koordinatensystem. Die tangentiale Beschleunigung/Verzögerung in Richtung der Bahn wird beschränkt.

Die Instruktion beschränkt nicht die Gesamtbeschleunigung der Ausrüstung, d. h. die Beschleunigung im Welt-Koordinatensystem, kann daher nicht direkt benutzt werden, um die Ausrüstung vor hohen Beschleunigungen zu schützen.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.



xx0500002184

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `PathAccLim`:

Siehe auch *Weitere Beispiele auf Seite 509*.

Beispiel 1:

```
PathAccLim TRUE \AccMax := 4, TRUE \DecelMax := 4;
```

TCP-Beschleunigung und TCP-Verzögerung sind auf 4 m/s ^{2}.

Beispiel 2:

```
PathAccLim FALSE, FALSE;
```

TCP-Beschleunigung und -Verzögerung sind auf das Maximum gesetzt (Standard).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.159 PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn

RobotWare - OS

Fortsetzung

Argumente

```
PathAccLim AccLim [\AccMax] DecelLim [\DecelMax]
```

`AccLim`

Datentyp: `bool`

TRUE, wenn die Beschleunigung beschränkt wird, sonst FALSE.

`[ \AccMax ]`

Datentyp: `num`

Der absolute Wert der Beschleunigungsbeschränkung in m/s ^{2}. Nur verwendet, wenn `AccLim` **TRUE ist.**

`DecelLim`

Datentyp: `bool`

TRUE, wenn die Verzögerung beschränkt wird, sonst FALSE.

`[ \DecelMax ]`

Datentyp: `num`

Der absolute Wert der Verzögerungsbeschränkung in m/s ^{2}. Nur verwendet, wenn `DecelLim` **TRUE ist.**

Programmabarbeitung

Die Beschleunigungs-/Verzögerungseinschränkungen gelten für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue `PathAccLim`**-Instruktion ausgeführt wird.**

Die maximale Beschleunigung/Verzögerung (`PathAccLim FALSE, FALSE`**) wird automatisch gesetzt**

- **bei Verwendung des Neustart-Modus RAPID zurücksetzen.**
- **wenn ein neues Programm oder ein neues Modul geladen wird**
- **bei Beginn der Programmabarbeitung vom Anfang**
- **beim Verschieben des Programmzeigers auf** `main`
- **wenn der Programmzeiger auf eine Routine verschoben wird**
- **wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.**

Bei Kombination der Instruktionen `AccSet` **und** `PathAccLim` **reduziert das System die Beschleunigung/Verzögerung in der folgenden Reihenfolge:**

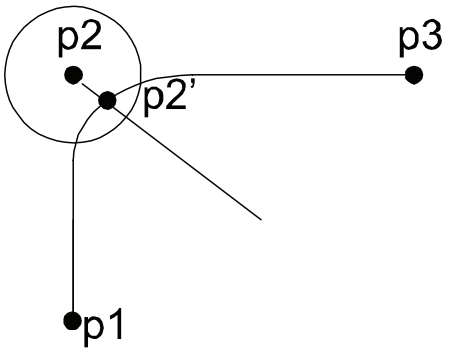
- **gemäß** `AccSet`
- **gemäß** `PathAccLim`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `PathAccLim` sind unten aufgeführt.



xx0500002183

Beispiel 1:

```
MoveL p1, v1000, fine, tool0;
PathAccLim TRUE\AccMax := 4, FALSE;
MoveL p2, v1000, z30, tool0;
MoveL p3, v1000, fine, tool0;
PathAccLim FALSE, FALSE;
```

TCP-Beschleunigung ist auf 4 m/s^2 zwischen `p1` und `p3` beschränkt.

Beispiel 2:

```
MoveL p1, v1000, fine, tool0;
MoveL p2, v1000, z30, tool0;
PathAccLim TRUE\AccMax :=3, TRUE\DecelMax := 4;
MoveL p3, v1000, fine, tool0;
PathAccLim FALSE, FALSE;
```

TCP-Beschleunigung ist auf 3 m/s^2 zwischen `p2'` und `p3` beschränkt.

TCP-Verzögerung ist auf 4 m/s^2 zwischen `p2'` und `p3` beschränkt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ACC_TOO_LOW`	Der Parameter `\AccMax` oder `\DecelMax` ist zu niedrig gesetzt.

Einschränkungen

Die minimal zulässige Beschleunigung/Verzögerung beträgt 0.1 m/s^2. Es wird empfohlen, die Beschränkungen für Beschleunigung und Verzögerung symmetrisch zu halten, das heißt dieselben Werte für `AccMax` und `DecelMax` zu verwenden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.159 PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn

RobotWare - OS

Fortsetzung

Syntax

```
PathAccLim
  [ AccLim ':=' ] < expression (IN) of bool >
  [ '\' AccMax ':=' <expression (IN) of num >] ','
  [ DecelLim ':=' ] < expression (IN) of bool>
  [ '\' DecelMax ':=' <expression (IN) of num >] ';'
```

Weitere Informationen

Information über	Siehe
Reduktion der Beschleunigung	*AccSet - Verringern der Beschleunigung auf Seite 25*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Steuern der Beschleunigung im Welt-Koordinatensystem	*WorldAccLim - Steuern der Beschleunigung im Welt-Koordinatensystem auf Seite 1112*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.160 PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück

Verwendung

`PathLengthReset` wird verwendet, um den Zähler zurückzusetzen, der die vom TCP des Roboters zurückgelegte Bahnlänge überwacht.

Diese Instruktion kann jederzeit aufgerufen werden, es ist jedoch ratsam, sie aufzurufen, wenn der Roboter stillsteht, um ein vorhersehbares Verhalten für den Zähler zu erhalten.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathLengthReset`.

Beispiel 1:

```
PathLengthStart;
MoveJ p10, v1000, z50, L10tip;
...
MoveL p40, v1000, fine, L10tip;
PathLengthStop;
TPWrite "PathLengthGet: "+ValToStr(PathLengthGet());
PathLengthReset;
```

In diesem Beispiel wird der Wert des Zählers ausgelesen, der die vom TCP des Roboters zurückgelegte Bahnlänge misst. Der Wert wird dann in den FlexPendant geschrieben.

Programmabarbeitung

Die Bahnlängenmessung gilt jeweils für die nächste abgearbeitete Roboterbewegungsinstruktion und ist so lange wirksam, bis eine `PathLengthStop`-Instruktion ausgeführt wird.

Die Bahnlängenmessung wird ausgeschaltet und der Bahnlängenmessungszähler wird auf null gesetzt, wenn ein Befehl `PathLengthReset` ausgeführt wird. Der Standardwert, Bahnlängenmessung aus, wird automatisch eingestellt:

- bei Verwendung des Neustart-Modus *RAPID zurücksetzen*.
- beim Laden eines neuen Programms oder eines neuen Moduls.
- und bei Beginn der Programmabarbeitung vom Anfang.
- beim Bewegen des Programmzeigers zu Hauptroutine.
- beim Bewegen des Programmzeiger zu einer Routine.
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

Bahnlängenmessung gelten nur für TCP Roboter.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.160 PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück

RobotWare - OS

Fortsetzung

Syntax

```
PathLengthReset';'
```

Weitere Informationen

Information über	Siehe
`PathLengthGet`	*PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus auf Seite 1427*
`PathLengthStart`	*PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht auf Seite 513*
`PathLengthStop`	*PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht auf Seite 515*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.161 PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht

Verwendung

`PathLengthStart` wird verwendet, um den Zähler zu aktivieren, der die vom TCP des Roboters zurückgelegte Bahnlänge ab dem nächsten Move-Instruktion überwacht. Die Bahnlänge wird immer im Werkobjekt gemessen. Beachten Sie, dass der interne Zähler für die TCP-Länge durch die Instruktion nicht zurückgesetzt wird. Zum Zurücksetzen des Zählers `PathLengthReset` verwendend.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathLengthStart`.

Beispiel 1:

```
PathLengthStart;
MoveJ p10, v1000, z50, L10tip;
...
MoveL p40, v1000, fine, L10tip;
PathLengthStop;
TPWrite "PathLengthGet: "+ValToStr(PathLengthGet());
PathLengthReset;
```

In diesem Beispiel wird der Wert des Zählers ausgelesen, der die vom TCP des Roboters zurückgelegte Bahnlänge misst. Der Wert wird dann in den FlexPendant geschrieben.

Programmabarbeitung

Die Bahnlängenmessung gilt jeweils für die nächste abgearbeitete Roboterbewegungsinstruktion und ist so lange wirksam, bis eine `PathLengthStop`-Instruktion ausgeführt wird.

Die Bahnlängenmessung wird ausgeschaltet und der Bahnlängenmessungszähler wird auf null gesetzt, wenn ein Befehl `PathLengthReset` ausgeführt wird. Der Standardwert, Bahnlängenmessung aus, wird automatisch eingestellt:

- bei Verwendung des Neustart-Modus *RAPID zurücksetzen*.
- beim Laden eines neuen Programms oder eines neuen Moduls.
- und bei Beginn der Programmabarbeitung vom Anfang.
- beim Bewegen des Programmzeigers zu Hauptroutine.
- beim Bewegen des Programmzeiger zu einer Routine.
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

Bahnlängenmessung gelten nur für TCP Roboter.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.161 PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht

RobotWare - OS

Fortsetzung

Syntax

```
PathLengthStart';'
```

Weitere Informationen

Information über	Siehe
`PathLengthReset`	*PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück auf Seite 511*
`PathLengthGet`	*PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus auf Seite 1427*
`PathLengthStop`	*PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht auf Seite 515*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.162 PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht

Verwendung

`PathLengthStop` wird verwendet, um den Zähler ab der nächsten Move-Instruktion zu stoppen, der die vom TCP des Roboters zurückgelegte Bahnlänge überwacht.

Die Bahnlänge wird immer im Werkobjekt gemessen. Beachten Sie, dass der interne Zähler für die TCP-Länge durch die Instruktion nicht zurückgesetzt wird. Zum Zurücksetzen des Zählers `PathLengthReset` verwenden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathLengthStop`.

Beispiel 1:

```
PathLengthStart;
MoveJ p10, v1000, z50, L10tip;
...
MoveL p40, v1000, fine, L10tip;
PathLengthStop;
TPWrite "PathLengthGet: "+ValToStr(PathLengthGet());
PathLengthReset;
```

In diesem Beispiel wird der Wert des Zählers ausgelesen, der die vom TCP des Roboters zurückgelegte Bahnlänge misst. Der Wert wird dann in den FlexPendant geschrieben.

Programmabarbeitung

Die Bahnlängenmessung gilt jeweils für die nächste abgearbeitete Roboterbewegungsinstruktion und ist so lange wirksam, bis eine `PathLengthStop`-Instruktion ausgeführt wird.

Die Bahnlängenmessung wird ausgeschaltet und der Bahnlängenmessungszähler wird auf null gesetzt, wenn ein Befehl `PathLengthReset` ausgeführt wird. Der Standardwert, Bahnlängenmessung aus, wird automatisch eingestellt:

- bei Verwendung des Neustart-Modus *RAPID zurücksetzen*.
- beim Laden eines neuen Programms oder eines neuen Moduls.
- und bei Beginn der Programmabarbeitung vom Anfang.
- beim Bewegen des Programmzeigers zu Hauptroutine.
- beim Bewegen des Programmzeiger zu einer Routine.
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

Bahnlängenmessung gelten nur für TCP Roboter.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.162 PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht

RobotWare - OS

Fortsetzung

Syntax

```
PathLengthStop';'
```

Weitere Informationen

Information über	Siehe
`PathLengthReset`	*PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück auf Seite 511*
`PathLengthGet`	*PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus auf Seite 1427*
`PathLengthStart`	*PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht auf Seite 513*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.163 PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen

Verwendung

Mit `PathRecMoveBwd` wird der Roboter auf einer aufgezeichneten Bahn zurückbewegt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathRecMoveBwd`:

Siehe auch *Weitere Beispiele auf Seite 518*.

Beispiel 1:

```
VAR pathrecid fixture_id;
PathRecMoveBwd \ID:=fixture_id \ToolOffs:=[0, 0, 10] \Speed:=v500;
```

Der Roboter wird zu der Position im Programm zurückbewegt, an der von der Instruktion `PathRecStart` der Bezeichner `fixture_id` gesetzt wurde. Der TCP-Offset beträgt 10 mm in Z-Richtung und die Geschwindigkeit ist auf 500 mm/s eingestellt.

Argumente

```
PathRecMoveBwd [\ID] [\ToolOffs] [\Speed]
```

`[\ID]`

Identifier

Datentyp: `pathrecid`

Diese Variable gibt das ID-Ziel für die Rückwärtsbewegung an. Der Datentyp `pathrecid` weist keinen Wert auf (Non-Value) und wird nur als Bezeichner für die Benennung der Aufzeichnungsposition verwendet.

Falls kein ID-Ziel angegeben ist, erfolgt die Rückwärtsbewegung in einem Einzelsystem zum nächstgelegenen bekannten ID-Ziel. In einem MultiMove-System im synchronisierten Modus erfolgt die Rückwärtsbewegung jedoch zum nächstgelegenen der folgenden Ziele:

- Zurück an die Position, an der die synchronisierte Bewegung begonnen hat.
- Zurück zum nächstgelegenen bekannten ID-Ziel

`[\ToolOffs]`

Tool Offset

Datentyp: `pos`

Gibt das Offset-Spiel für den TCP während der Bewegung an. Für die TCP-Koordinaten wird ein kartesisches Offset-Koordinatensystem verwendet. Ein positiver Z-Offset zeigt ein Spiel an. Das ist hilfreich, wenn der Roboter einen Prozess ausführt, bei dem Material hinzugefügt wird. Bei synchronisierten Bewegungen muss dieses Argument von allen oder von keiner der mechanischen Einheiten verwendet werden. Falls für einige der mechanischen Einheiten kein Offset erwünscht ist, kann ein Null-Offset verwendet werden. Auch mechanische Einheiten ohne TCP müssen dieses Argument verwenden, falls in einer anderen Task ein Roboter mit TCP verwendet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.163 PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen

Path Recovery
Fortsetzung

`[\Speed]`

Datentyp: `speeddata`

Die Geschwindigkeit ersetzt die ursprünglich für die Vorwärtsbewegung verwendete Geschwindigkeit. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen. Falls vorhanden, wird diese Geschwindigkeit für die gesamte Rückwärtsbewegung verwendet. Falls nicht vorhanden, wird die Rückwärtsbewegung mit der Geschwindigkeit der ursprünglichen Bewegungsinstruktion ausgeführt.

Programmabarbeitung

Die Bahnaufzeichnung wird mit der Instruktion `PathRecStart` aktiviert. Nach dem Start der Aufzeichnung werden alle Bewegungsinstruktionen aufgezeichnet und der Roboter kann sich entlang der aufgezeichneten Bahn mithilfe von `PathRecMoveBwd` zu jedem Punkt zurückbewegen.

Synchronisierte Bewegung

Bei der Ausführung der Bahnaufzeichnung bei synchronisierter Bewegung sind ein paar Dinge zu berücksichtigen.

- Alle mit der aufgezeichneten synchronen Bewegung in Zusammenhang stehenden Tasks müssen `PathRecMoveBwd` anfordern, bevor einer der Roboter mit der Bewegung beginnt.
- Die komplette Synchronisierung wird in umgekehrter Richtung aufgezeichnet und ausgeführt. Wenn beispielsweise `PathRecMoveBwd` aus einem synchronisierten Block heraus an eine unabhängige Position angefordert wird, ändert die Bahnaufzeichnung den Status in der Instruktion `SyncMoveOn` automatisch in „Unabhängig“.
- `SyncMoveOn` wird ohne Bahnbezeichner als Stopppunkt betrachtet. Das bedeutet, wenn die Bahnaufzeichnung mithilfe von `PathRecStart` gestartet wurde und `PathRecMoveBwd` ohne das optionale Argument `\ID` ausgeführt wird, bewegt sich der Roboter bis zu der Position zurück, an der er sich bei der Ausführung von `SyncMoveOn` befand. Da die Rückwärtsbewegung vor `SyncMoveOn` gestoppt wird, wird der Status in „Unabhängig“ geändert.
- `WaitSyncTask` wird ohne Bahnbezeichner als Stopppunkt betrachtet. Das bedeutet, wenn die Bahnaufzeichnung mithilfe von `PathRecStart` gestartet wurde und `PathRecMoveBwd` ausgeführt wird, bewegt sich der Roboter nicht weiter als bis zu der Position zurück, an der er sich bei der Ausführung von `WaitSyncTask` befand.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `PathRecMoveBwd` sind unten aufgeführt.

Beispiel 1 - Unabhängige Bewegung

```
VAR pathrecid safe_id;
CONST robtarget p0 := [...];
...
CONST robtarget p4 := [...];
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
VAR num choice;

MoveJ p0, vmax, z50, tool1;
PathRecStart safe_id;
MoveJ p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
MoveL p3, vmax, z50, tool1;
MoveL p4, vmax, z50, tool1;

ERROR:
TPReadFK choice,"Go to safe?",stEmpty,stEmpty,stEmpty,stEmpty,"Yes";
  IF choice=5 THEN
    IF PathRecValidBwd(\ID:=safe_id) THEN
      StorePath;
      PathRecMoveBwd \ID:=safe_id \ToolOffs:=[0, 0 , 10];
      Stop;
      !Fix problem
      PathRecMoveFwd;
      RestoPath;
      StartMove;
      RETRY;
    ENDIF
  ENDIF
```

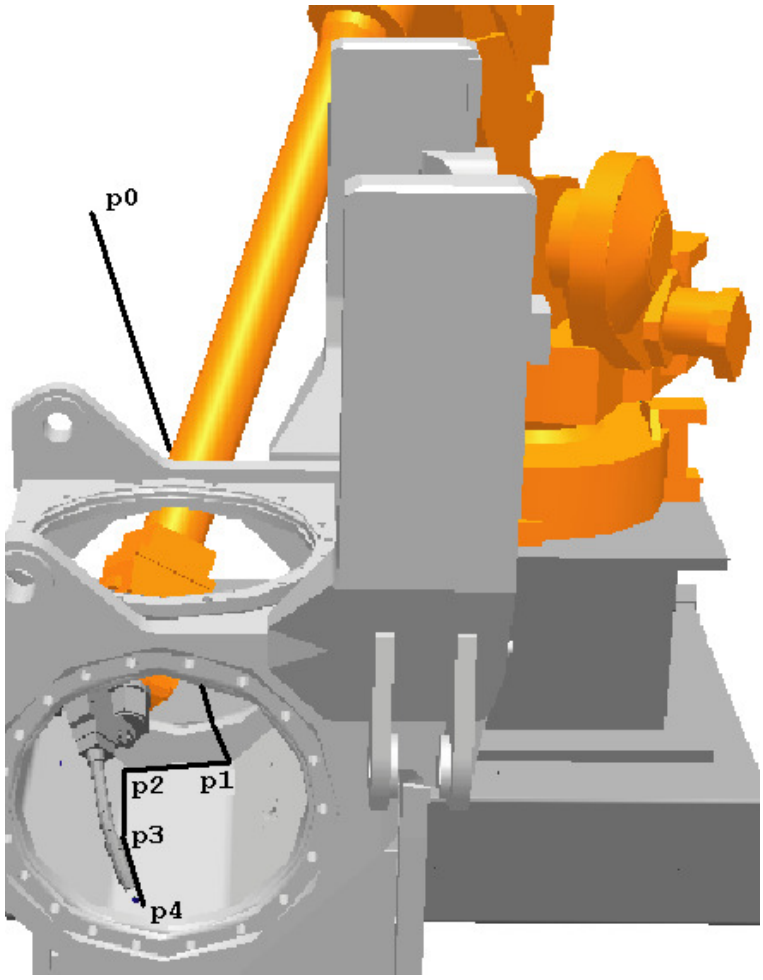
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.163 PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen

Path Recovery

Fortsetzung



xx0500002135

Dieses Beispiel zeigt, wie die Bahnaufzeichnung verwendet werden kann, um den Roboter bei Fehlern aus engen Stellen herauszubewegen, ohne eine bestimmte Bahn programmieren zu müssen.

Ein Bauteil wird angefertigt. Am Näherungspunkt `p0` wird die Bahnaufzeichnung gestartet und der Bezeichner `safe_id` der Bahnaufzeichnung wird gesetzt. Angenommen, bei der Bewegung des Roboters von `p3` nach `p4` tritt ein behebbarer Fehler auf. An diesem Punkt wird die Bahn mithilfe von `StorePath` gespeichert. Durch das Speichern der Bahn kann die Fehlerbehandlung mit einer neuen Bewegung beginnen und später die ursprüngliche Bewegungsbahn wieder aufnehmen. Nach dem Speichern der Bahn kann der Roboter mithilfe der Bahnaufzeichnung an eine sichere Position bewegt werden (`p0`), indem die Instruktion `PathRecMoveBwd` ausgeführt wird.

Beachten Sie, dass für das Werkzeug ein Offset eingeplant wird, um ein gewisses Spiel zur Verfügung zu stellen (z. B. für neue Schweißnähte). Wenn der Roboter wegbewegt wurde, kann der Bediener die zur Fehlerbehandlung erforderlichen Schritte durchführen (z. B. den Schweißbrenner reinigen). Anschließend wird der Roboter mithilfe der Instruktion `PathRecMoveFwd` an die Stelle zurückbewegt, an der der Fehler aufgetreten ist. An dieser Position wird die Bahnebene mithilfe von

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`RestoPath` wieder auf die Basisebene umgestellt und es wird ein neuer Versuch unternommen.

Beispiel 2 - Synchronisierte Bewegung

T_ROB1

```
VAR pathrecid HomeROB1;
CONST robtarget pR1_10:=[...];
...
CONST robtarget pR1_60:=[...];

PathRecStart HomeROB1;
MoveJ pR1_10, v1000, z50, tGun;
MoveJ pR1_20, v1000, z50, tGun;
MoveJ pR1_30, v1000, z50, tGun;
SyncMoveOn sync1, tasklist;
MoveL pR1_40 \ID:=1, v1000, z50, tGun\wobj:=pos1;
MoveL pR1_50 \ID:=2, v1000, z50, tGun\wobj:=pos1;
MoveL pR1_60 \ID:=3, v1000, z50, tGun\wobj:=pos1;
SyncMoveOff sync2;

ERROR
  StorePath \KeepSync;
  TEST ERRNO
  CASE ERR_PATH_STOP:
    PathRecMoveBwd \ID:= HomeROB1\ToolOffs:=[0,0,10];
  ENDTEST
  !Perform service action
  PathRecMoveFwd \ToolOffs:=[0,0,10];
  RestoPath;
  StartMove;
```

T_ROB2

```
VAR pathrecid HomeROB2;
CONST robtarget pR2_10:=[...];
...
CONST robtarget pR2_50:=[...];

PathRecStart HomeROB2;
MoveJ pR2_10, v1000, z50, tGun;
MoveJ pR2_20, v1000, z50, tGun;
SyncMoveOn sync1, tasklist;
MoveL pR2_30 \ID:=1, v1000, z50, tGun\wobj:=pos1;
MoveL pR2_40 \ID:=2, v1000, z50, tGun\wobj:=pos1;
MoveL pR2_50 \ID:=3, v1000, z50, tGun\wobj:=pos1;
SyncMoveOff sync2;

ERROR
  StorePath \KeepSync;
  TEST ERRNO
  CASE ERR_PATH_STOP:
    PathRecMoveBwd \ToolOffs:=[0,0,10];
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.163 PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen
Path Recovery
Fortsetzung

```
  ENDTEST
  !Perform service action
  PathRecMoveFwd \ToolOffs:=[0,0,10];
  RestoPath;
  StartMove;
```

T_ROB3

```
VAR pathrecid HomePOS1;
CONST jointtarget jP1_10:=[...];
...
CONST jointtarget jP1_40:=[...];

PathRecStart HomePOS1;
MoveExtJ jP1_10, v1000, z50;
SyncMoveOn sync1, tasklist;
MoveExtJ jP1_20 \ID:=1, v1000, z50;
MoveExtJ jP1_30 \ID:=2, v1000, z50;
MoveExtJ jP1_40 \ID:=3, v1000, z50;
SyncMoveOff sync2;

ERROR
  StorePath \KeepSync;
  TEST ERRNO
  CASE ERR_PATH_STOP:
    PathRecMoveBwd \ToolOffs:=[0,0,0];
  DEFAULT:
  PathRecMoveBwd \ID:=HomePOS1\ToolOffs:=[0,0,0];
  ENDTEST
  !Perform service action
  PathRecMoveFwd \ToolOffs:=[0,0,0];
  RestoPath;
  StartMove;
```

Ein System besteht aus drei Manipulatoren, von denen jeder in einer separaten Task ausgeführt wird. Angenommen, für `T_ROB1` tritt im synchronisierten Block `sync1` der Fehler `ERR_PATH_STOP` auf. Bei einem Fehler soll der Roboter in die mit dem Bahnbezeichner `HomeROB1` bezeichnete Grundstellung zurückbewegt werden, um eine Wartung der externen Geräte des Roboters durchzuführen. Dies geschieht mithilfe der Instruktion `PathRecMoveBwd` und dem Bezeichner `pathrecid`.

Da der Fehler während der synchronisierten Bewegung aufgetreten ist, müssen der zweite Roboter mit TCP `T_ROB2` und die externe Achse `T_POS1` ebenfalls `PathRecMoveBwd` anfordern. Diese Manipulatoren müssen nicht weiter als vor den Startpunkt der synchronisierten Bewegung zurückbewegt werden. Indem `PathRecMoveBwd` bei `ERR_PATH_STOP` nicht mit einem Bahnbezeichner bereitgestellt wird, wird die Fähigkeit der Bahnaufzeichnung genutzt, nach `SyncMoveOn` anzuhalten. Beachten Sie, dass die externe Achse ohne TCP über einen Null-Offset verfügt, um den Robotern mit TCP diese Möglichkeit zu eröffnen.

Das `DEFAULT`-Verhalten in der `ERROR`-Behandlungsroutine in diesem Beispiel ist, dass alle Manipulatoren zunächst die synchronisierten Bewegungen und

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

anschließend die unabhängigen Bewegungen zurück an den Startpunkt der aufgezeichneten Bahn ausführen. Dies wird durch Angabe von `\ID` in `PathRecMoveBwd` für alle Manipulatoren erreicht.

Einschränkungen

Bewegungen mithilfe der Bahnaufzeichnung können nicht auf Basisebene durchgeführt werden, d. h., `StorePath` muss vor `PathRecMoveBwd` ausgeführt werden.

Eine Rückwärtsbewegung über eine `SynchMoveOff`-Anweisung ist nicht möglich.

Eine Rückwärtsbewegung über eine `WaitSyncTask`-Anweisung ist nicht möglich.

`SyncMoveOn` muss mindestens eine unabhängige Bewegung vorangehen, wenn eine Bewegung zurück an die Position möglich sein soll, an der die synchronisierte Bewegung begonnen hat.

Wenn nicht zu dem Punkt zurückgekehrt werden soll, an dem `PathRecMoveBwd` ausgeführt wurde (mithilfe von `PathRecMoveFwd`), muss die Bahnaufzeichnung mithilfe von `PathRecStop` gestoppt werden. Mit `PathRecStop\Clear` wird zudem die aufgezeichnete Bahn gelöscht.

`PathRecMoveBwd` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart,Reset oder Step.

Syntax

```
PathRecMoveBwd
  [ '\' ID ':=' < variable (VAR) of pathrecid > ]
  [ '\' ToolOffs ':=' <expression (IN) of pos> ]
  [ '\' Speed ':=' <expression (IN) of speeddata> ]';'
```

Weitere Informationen

Information über	Siehe
Bezeichner der Bahnaufzeichnung	*pathrecid - Bezeichner der Bahnaufzeichnung auf Seite 1753*
Starten/Stoppen der Bahnaufzeichnung	*PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *PathRecStop - Stoppen der Bahnaufzeichnung auf Seite 531*
Prüfung auf gültige aufgezeichnete Bahn	*PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde auf Seite 1431* *PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde auf Seite 1434*
Vorwärtsbewegen der Bahnaufzeichnung	*PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen auf Seite 525*
Speichern/Wiederherstellen von Bahnen	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.163 PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen

Path Recovery

Fortsetzung

Information über	Siehe
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.164 PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen

Verwendung

`PathRecMoveFwd` wird verwendet, um den Roboter zurück an die Position zu bewegen, an der `PathRecMoveBwd` abgearbeitet wurde. Außerdem kann der Roboter stückweise vorwärts bewegt werden, indem ein Bezeichner bereitgestellt wird, der während der Rückwärtsbewegung übergeben wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathRecMoveFwd`:

Siehe auch *Weitere Beispiele auf Seite 526*.

Beispiel 1:

```
PathRecMoveFwd;
```

Der Roboter wird an die Position zurückbewegt, an der die Bahnaufzeichnung mit der Rückwärtsbewegung begonnen hatte.

Argumente

```
PathRecMoveFwd [\ID] [\ToolOffs] [\Speed]
```

[\ID]

Identifier

Datentyp: `pathrecid`

Diese Variable gibt das ID-Ziel für die Vorwärtsbewegung an. Der Datentyp `pathrecid` weist keinen Wert auf (Non-Value) und wird nur als Bezeichner für die Benennung der Aufzeichnungsposition verwendet.

Falls kein ID-Ziel angegeben ist, erfolgt die Vorwärtsbewegung immer bis zur Interrupt-Position auf der ursprünglichen Bahn.

[\ToolOffs]

Tool Offset

Datentyp: `pos`

Gibt das Offset-Spiel für den TCP während der Bewegung an. Für die TCP-Koordinaten wird ein kartesisches Koordinatensystem verwendet. Das ist hilfreich, wenn der Roboter einen Prozess ausführt, bei dem Material hinzugefügt wird.

`[\Speed]`

Datentyp: `speeddata`

\Speed überschreibt die ursprünglich für die Vorwärtsbewegung verwendete Geschwindigkeit. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen. Falls vorhanden, wird diese Geschwindigkeit für die gesamte Vorwärtsbewegung verwendet. Falls nicht vorhanden, wird die Vorwärtsbewegung mit der Geschwindigkeit der ursprünglichen Bewegungsinstruktion ausgeführt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.164 PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen

PathRecovery

Fortsetzung

Programmabarbeitung

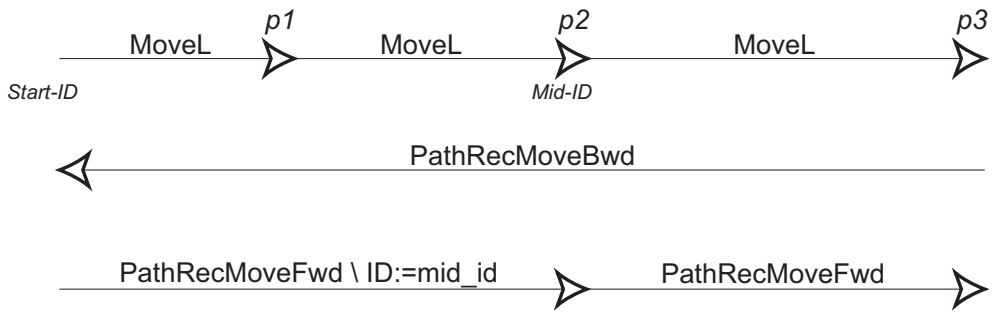
Die Bahnaufzeichnung wird mit der Instruktion `PathRecStart` aktiviert. Nach dem Start der Aufzeichnung kann der Roboter auf der ausgeführten Bahn zurückbewegt werden, indem `PathRecMoveBwd` ausgeführt wird. Im Anschluss kann der Roboter wieder an die Position zurückbewegt werden, an der mit der Rückwärtsbewegung begonnen wurde, indem `PathRecMoveFwd` ausgeführt wird. Außerdem kann der Roboter stückweise vorwärts bewegt werden, indem ein Bezeichner bereitgestellt wird, der während der Rückwärtsbewegung übergeben wurde.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `PathRecMoveFwd` sind unten aufgeführt.

```
VAR pathrecid start_id;
VAR pathrecid mid_id;
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];

  PathRecStart start_id;
  MoveL p1, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStart mid_id;
  MoveL p3, vmax, z50, tool1;
  StorePath;
  PathRecMoveBwd \ID:=start_id;
  PathRecMoveFwd \ID:=mid_id;
  PathRecMoveFwd;
  RestoPath;
```



xx0500002133

Im Beispiel oben wird die Bahnaufzeichnung gestartet und der Startpunkt wird mit dem Bahnbezeichner `start_id` versehen. Danach wird der Roboter mit den herkömmlichen Bewegungsinstruktionen nach vorne und dann entlang der aufgezeichneten Bahn wieder zum Bezeichner `start_id` der Bahnaufzeichnung zurückbewegt. Schließlich wird er mithilfe der Instruktion `PathRecMoveFwd` in zwei Schritten wieder vorwärts bewegt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Bewegungen mithilfe der Bahnaufzeichnung müssen auf Interrupt-Ebene durchgeführt werden, d. h. `StorePath` muss vor `PathRecMoveFwd` ausgeführt werden.

Um `PathRecMoveFwd` ausführen zu können, muss vorher die Instruktion `PathRecMoveBwd` ausgeführt worden sein.

Wenn nicht zu dem Punkt zurückgekehrt werden soll, an dem `PathRecMoveBwd` ausgeführt wurde (mithilfe von `PathRecMoveFwd`), muss die Bahnaufzeichnung mithilfe von `PathRecStop` gestoppt werden. Mit `PathRecStop\Clear` wird zudem die aufgezeichnete Bahn gelöscht.

`PathRecMoveFwd` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
PathRecMoveFwd '('
  [ '\' ID ':=' < variable (VAR) of pathid > ]
  [ '\' ToolOffs ':=' <expression (IN) of pos> ]
  [ '\' Speed ':=' <expression (IN) of speeddata> ]';'
```

Weitere Informationen

Information über	Siehe
Bezeichner der Bahnaufzeichnung	*pathrecid - Bezeichner der Bahnaufzeichnung auf Seite 1753*
Starten/Stoppen der Bahnaufzeichnung	*PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *PathRecStop - Stoppen der Bahnaufzeichnung auf Seite 531*
Prüfung auf gültige aufgezeichnete Bahn	*PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde auf Seite 1431* *PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde auf Seite 1434*
Bahnaufzeichnung rückwärts ablaufen lassen	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*
Speichern/Wiederherstellen von Bahnen	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview* *Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.165 PathRecStart - Starten der Bahnaufzeichnung

Verwendung

`PathRecStart` wird verwendet, um die Aufzeichnung der Bahn des Roboters zu starten. Die Bahninformationen werden von der Bahnaufzeichnung während der Ausführung eines RAPID-Programms aufgezeichnet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathRecStart`:

Beispiel 1:

```
VAR pathrecid fixture_id;

PathRecStart fixture_id;
```

Die Bahnaufzeichnung beginnt und der Startpunkt (die Position der Instruktion im RAPID-Programm) wird mit dem Bezeichner `fixture_id` gekennzeichnet.

Argumente

```
PathRecStart ID
```

`ID`

Identifier

Datentyp:`pathrecid`

Variable, die den Namen der Aufzeichnungsstartposition angibt. Der Datentyp `pathrecid` weist keinen Wert auf (Non-Value) und wird nur als Bezeichner für die Benennung der Aufzeichnungsposition verwendet.

Programmabarbeitung

Wenn die Bahnaufzeichnung gestartet wird, wird die Roboterbahn intern in der Robotersteuerung aufgezeichnet. Die aufgezeichnete Sequenz der Programmpositionen kann mithilfe der Instruktion `PathRecMoveBwd` rückwärts abgefahren werden. Der Roboter bewegt sich dann auf seiner zuvor ausgeführten Bahn rückwärts.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `PathRecStart` sind unten aufgeführt.

Beispiel 1:

```
VAR pathrecid origin_id;
VAR pathrecid corner_id;
VAR num choice;
MoveJ p1, vmax, z50, tool1;
PathRecStart origin_id;
MoveJ p2, vmax, z50, tool1;
PathRecStart corner_id;
MoveL p3, vmax, z50, tool1;
MoveAbsJ jt4, vmax, fine, tool1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
ERROR
  TPReadFK choice,"Extract
       to:",stEmpty,stEmpty,stEmpty,"Origin","Corner";
  IF choice=4 OR choice=5 THEN
    StorePath;
    IF choice=4 THEN
      PathRecMoveBwd \ID:=origin_id;
    ELSE
      PathRecMoveBwd \ID:=corner_id;
    ENDIF
    Stop;
    !Fix problem
    PathRecMoveFwd;
    RestoPath;
    StartMove;
    RETRY;
  ENDIF
```

In obigem Beispiel wird die Bahnaufzeichnung verwendet, um den Roboter an eine Serviceposition zu bewegen, wenn im Normalbetrieb ein Fehler auftritt.

Der Roboter bewegt sich entlang einer Bahn. Nach Position `p1` wird die Bahnaufzeichnung gestartet. Nach Position `p2` wird ein weiterer Bahnbezeichner eingefügt. Angenommen, bei der Bewegung von Position `p3` nach Position `jt4` tritt ein behebbarer Fehler auf. Die Fehlerbehandlung wird aufgerufen und der Anwender kann wählen, ob der Roboter an die Position `Origin` (Punkt `p1`) oder an die Position `Corner` (Punkt `p2`) bewegt werden soll. Anschließend wird mithilfe von `StorePath` die Bahnebene gewechselt, damit später an der Fehlerposition neu gestartet werden kann. Wenn sich der Roboter von der Fehlerposition wegbewegt hat, kann der Anwender den Fehler beheben (normalerweise eine Reparatur an den den Roboter umgebenden Geräten).

Anschließend wird der Roboter an die Position zurückbewegt, an der der Fehler aufgetreten ist. Die Bahnebene wird wieder in den Normalmodus versetzt und es wird ein neuer Versuch unternommen.

Einschränkungen

Die Bahnaufzeichnung kann nur auf der Basisebene der Bahn gestartet werden, d. h., Bewegungen auf `StorePath`-Ebene werden nicht aufgezeichnet.

Syntax

```
PathRecStart
  [ ID ':='] < variable (VAR) of pathrecid> ';'
```

Weitere Informationen

Information über	Siehe
Bezeichner der Bahnaufzeichnung	*pathrecid - Bezeichner der Bahnaufzeichnung auf Seite 1753*
Beenden der Bahnaufzeichnung	*PathRecStop - Stoppen der Bahnaufzeichnung auf Seite 531*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.165 PathRecStart - Starten der Bahnaufzeichnung

Path Recovery

Fortsetzung

Information über	Siehe
Prüfung auf gültige aufgezeichnete Bahn	*PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde auf Seite 1431* *PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde auf Seite 1434*
Abspielen der Bahnaufzeichnung rückwärts	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*
Abspielen der Bahnaufzeichnung vorwärts	*PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen auf Seite 525*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.166 PathRecStop - Stoppen der Bahnaufzeichnung

Verwendung

`PathRecStop` wird verwendet, um die Aufzeichnung der Bahn des Roboters anzuhalten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathRecStop`:

Siehe auch *Weitere Beispiele* weiter unten.

Beispiel 1:

```
PathRecStop \Clear;
```

Die Bahnaufzeichnung wird gestoppt und der Pufferspeicher mit den gespeicherten Bahninformationen wird gelöscht.

Argumente

```
PathRecStop [\Clear]
```

`[\Clear]`

Datentyp: `switch`

Die aufgezeichnete Bahn wird gelöscht.

Programmabarbeitung

Wenn die Bahnaufzeichnung gestoppt wird, wird die Aufzeichnung der Bahn unterbrochen. Das optionale Argument `\Clear` löscht den Pufferspeicher mit den Bahninformationen und verhindert dadurch, dass die aufgezeichnete Bahn versehentlich ausgeführt wird.

Nach dem Stoppen der Bahnaufzeichnung mit `PathRecStop` können frühere Bahnaufzeichnungen nicht mehr für Rückwärtsbewegungen (`PathRecMoveBwd`) verwendet werden. Frühere Aufzeichnungen können jedoch verwendet werden, wenn `PathRecStart` von derselben Position aus aufgerufen wird, an der die Bahnaufzeichnung gestoppt wurde. Siehe das folgende Beispiel.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `PathRecStop` sind unten aufgeführt.

```
LOCAL VAR pathrecid id1;
LOCAL VAR pathrecid id2;
LOCAL CONST robtarget p0:= [...];
......
LOCAL CONST robtarget p6 := [...];
PROC example1()
  MoveL p0, vmax, z50, tool1;
  PathRecStart id1;
  MoveL p1, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStop;
  MoveL p3, vmax, z50, tool1;
```

Fortsetzung auf nächster Seite

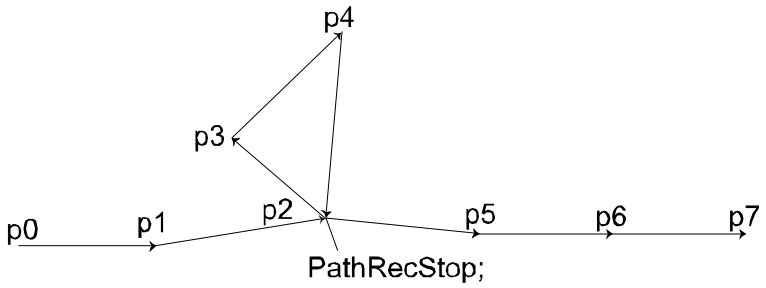
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.166 PathRecStop - Stoppen der Bahnaufzeichnung

Path Recovery

Fortsetzung

```
  MoveL p4, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStart id2;
  MoveL p5, vmax, z50, tool1;
  MoveL p6, vmax, z50, tool1;
  StorePath;
  PathRecMoveBwd \ID:=id1;
  PathRecMoveFwd;
  RestoPath;
  StartMove;
  MoveL p7, vmax, z50, tool1;
ENDPROC
PROC example2()
  MoveL p0, vmax, z50, tool1;
  PathRecStart id1;
  MoveL p1, vmax, z50, tool1;
  MoveL p2, vmax, z50, tool1;
  PathRecStop;
  MoveL p3, vmax, z50, tool1;
  MoveL p4, vmax, z50, tool1;
  PathRecStart id2;
  MoveL p2, vmax, z50, tool1;
  MoveL p5, vmax, z50, tool1;
  MoveL p6, vmax, z50, tool1;
  StorePath;
  PathRecMoveBwd \ID:=id1;
  PathRecMoveFwd;
  RestoPath;
  StartMove;
  MoveL p7, vmax, z50, tool1;
ENDPROC
```



PathRecStop_

Obige Beispiele beschreiben die Aufzeichnung der Roboterbahn, wenn die Aufzeichnung mitten in der Sequenz gestoppt wurde. In `example1` ist der Befehl `PathRecMoveBwd \ID:=id1;` gültig und der Roboter führt folgende Bahn aus: `p6 -> p5 -> p2 -> p1 -> p0`

Der Befehl ist gültig, weil die Bahnaufzeichnung an genau der gleichen Roboterposition gestoppt und wieder gestartet wurde. Wenn dies nicht erwünscht ist, sollte der Stoppbefehl das optionale Argument `\Clear` enthalten. Auf diese

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weise wird die aufgezeichnete Bahn gelöscht und es ist nicht mehr möglich, die vorherigen Bezeichner der Bahnaufzeichnung wiederherzustellen.

Der einzige Unterschied in `example2` besteht darin, dass die Aufzeichnung zum zweiten Mal gestartet wurde. In diesem Fall löst `PathRecMoveBwd \ID:=id1` eine Fehlermeldung aus. Der Grund hierfür ist, dass zwischen `p4`, `p3` und `p2` keine Bahn aufgezeichnet wurde. Es können `PathRecMoveBwd \ID:=id2` abgearbeitet werden.

Syntax

```
PathRecStop
  [ '\'switch Clear ] ';'
```

Weitere Informationen

Information über	Siehe
Bezeichner der Bahnaufzeichnung	*pathrecid - Bezeichner der Bahnaufzeichnung auf Seite 1753*
Starten der Bahnaufzeichnung	*PathRecStart - Starten der Bahnaufzeichnung auf Seite 528*
Prüfung auf gültige aufgezeichnete Bahn	*PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde auf Seite 1431* *PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde auf Seite 1434*
Abspielen der Aufzeichnung rückwärts	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*
Abspielen der Aufzeichnung vorwärts	*PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen auf Seite 525*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.167 PathResol - Override der Bahnauflösung

Verwendung

`PathResol` (*Path Resolution*) wird verwendet, um die konfigurierte geometrische Bahn-Abtastrate zu überschreiben, die in den Systemparametern für die von der aktuellen Programmtask gesteuerten mechanischen Einheiten definiert ist.

Die Abtastrate der geometrischen Bahn wird in Anwendungen mit Sensoreingängen verwendet. Beachten Sie, dass sie nicht für die Definition der geometrischen Bahn in der normalen Anwendung, eine Legacy-Funktionalität, verwendet wird.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Beschreibung

Beispiel für die Verwendung von `PathResol`:

- Verwendung der koordinierten Interpolation
- Verwendung von *Weldguide*.
- Verwendung der Option *Conveyor Tracking*.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PathResol`:

```
MoveJ p1,v1000,fine,tool1;
PathResol 150;
```

Wenn der Roboter an einem Stopppunkt steht, wird die Bahn-Abtastrate auf `150` % des konfigurierten Werts erhöht.

Argumente

```
PathResol PathSampleTime
```

`PathSampleTime`

Datentyp: `num`

Override als Prozentsatz der konfigurierten Bahn-Abtastrate. 100 % entspricht der konfigurierten Bahn-Abtastrate. Wertebereich von 25 bis 400 %.

Programmabarbeitung

Die Bahnauflösung gilt für alle nachfolgenden Positionierinstruktionen, bis eine neue `PathResol`-Instruktion abgearbeitet wird. Dies beeinflusst die Bahnauflösung bei der gesamten Programmabarbeitung von Bewegungen (Standardbahn und Bahn nach `StorePath`) sowie beim manuellen Bewegen.

Im MultiMove-System sind im synchronisierten, koordinierten Modus die folgenden Punkte gültig:

- Alle mechanischen Einheiten im synchronisierten koordinierten Modus werden mit der aktuellen Bahnauflösung für die tatsächlich verwendete Bewegungsplanung ausgeführt.
- Ein neuer Bahnauflösungsbefehl für die tatsächliche Bewegungsplanung wirkt sich auf die synchronisierte koordinierte Bewegung und zukünftige unabhängige Bewegungen in der Bewegungsplanung aus.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Ein neuer Bahnauflösungsbefehl für eine andere Bewegungsplanung wirkt sich nur auf zukünftige unabhängige Bewegungen in dieser Bewegungsplanung aus.

Informationen über die Verbindung zwischen Programmtask und Bewegungsplanung finden Sie im *Anwendungshandbuch - MultiMove*.

Der Standardwert für den Override der Bahn-Abtastrate beträgt 100 %. Dieser Wert wird automatisch gesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Der aktuelle Override der Bahn-Abtastrate kann aus der Variable `C_MOTSET` (Datentyp `motsetdata`) in der Komponente `pathresol` ausgelesen werden.

Einschränkung

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`PathResol` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop,Restart oder Step.

Syntax

```
PathResol
  [PathSampleTime ':='] <expression (IN) of num>';'
```

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungseinstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration der Bahnauflösung	*Technisches Referenzhandbuch - Systemparameter*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.168 PDispOff - Deaktivieren der Programmverschiebung

Verwendung

`PDispOff` (*Program Displacement Off*) deaktiviert eine Programmverschiebung.

Programmverschiebung wird durch die Instruktion `PDispSet` oder `PDispOn` aktiviert und gilt für alle Bewegungen, bis eine andere Programmverschiebung aktiviert wird oder bis die Programmverschiebung deaktiviert wird.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `PDispOff`:

Beispiel 1:

```
PDispOff;
```

Deaktiviert die Programmverschiebung.

Beispiel 2:

```
MoveL p10, v500, z10, tool1;
PDispOn \ExeP:=p10, p11, tool1;
MoveL p20, v500, z10, tool1;
MoveL p30, v500, z10, tool1;
PDispOff;
MoveL p40, v500, z10, tool1;
```

Eine Programmverschiebung wird als Differenz zwischen den Positionen `p10` und `p11` definiert. Diese Verschiebung beeinflusst die Bewegung zu `p20` und `p30`, aber nicht zu `p40`.

Programmabarbeitung

Die aktive Programmverschiebung wird zurückgesetzt. Dies bedeutet, dass das Programmverschiebungs-Koordinatensystem mit dem Objekt-Koordinatensystem übereinstimmt und sich damit alle programmierten Positionen auf das letztgenannte beziehen.

Syntax

```
PDispOff ';'
```

Weitere Informationen

Information über	Siehe
Definition einer Programmverschiebung mit zwei Positionen	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Definition einer Programmverschiebung mithilfe eines bekannten Koordinatensystems	*PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem auf Seite 542*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.169 PDispOn - Aktivieren der Programmverschiebung

Verwendung

`PDispOn` (*Program Displacement On*) wird verwendet, um eine Programmverschiebung mithilfe von zwei Roboterpositionen zu definieren und zu aktivieren.

Eine Programmverschiebung ist z. B. sinnvoll, nachdem eine Suche durchgeführt wurde oder wenn ähnliche Bewegungsmuster an mehreren verschiedenen Stellen im Programm verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `PDispOn`:

Siehe auch *Weitere Beispiele auf Seite 539*.

Beispiel 1:

```
MoveL p10, v500, z10, tool1;
PDispOn \ExeP:=p10, p20, tool1;
```

Aktiviert eine Programmverschiebung (parallele Bewegung). Dies wird anhand der Differenz zwischen den Positionen `p10` und `p20` berechnet.

Beispiel 2:

```
MoveL p10, v500, fine \Inpos := inpos50, tool1;
PDispOn *, tool1;
```

Aktiviert eine Programmverschiebung (parallele Bewegung). Da in der vorherigen Instruktion ein exakt definierter Stopppunkt verwendet wurde, kann auf das Argument `\ExeP` verzichtet werden. Die Verschiebung wird anhand der Differenz zwischen der tatsächlichen Position des Roboters und dem programmierten Punkt (*) berechnet, der in der Instruktion gespeichert wird.

Beispiel 3

```
PDispOn \Rot \ExeP:=p10, p20, tool1;
```

Aktiviert eine Programmverschiebung einschließlich Rotation. Dies wird anhand der Differenz zwischen den Positionen `p10` und `p20` berechnet.

Argumente

```
PDispOn [\Rot] [\ExeP] ProgPoint Tool [\WObj]
```

`[ \Rot ]`

Rotation

Datentyp: `switch`

Die Differenz in der Werkzeugorientierung wird berücksichtigt. Dies erfordert ein Drehen des Programms.

`[ \ExeP ]`

Executed Point

Datentyp: `robtarget`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.169 PDispOn - Aktivieren der Programmverschiebung
RobotWare - OS
Fortsetzung

Die neue Roboterposition, die zur Berechnung der Verschiebung verwendet wird. Wenn das Argument weggelassen wird, wird die aktuelle Roboterposition zum Zeitpunkt der Programmabarbeitung verwendet.

`ProgPoint`

Programmed Point

Datentyp: `robtarget`

Die ursprüngliche Position des Roboters zum Zeitpunkt der Programmierung.

`Tool`

Datentyp: `tooldata`

Das bei der Programmierung verwendete Werkzeug, d. h. der TCP, auf den sich die Position des `ProgPoint` bezieht.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

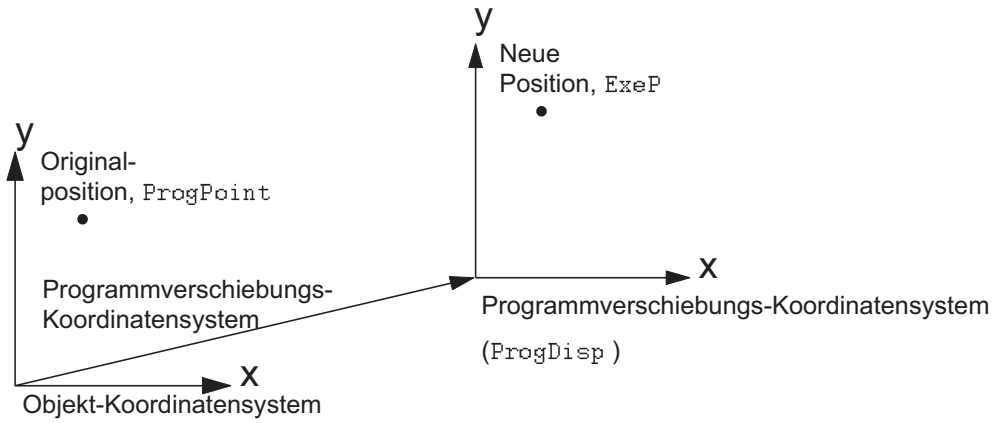
Das Werkobjekt (Koordinatensystem), auf das sich die Position `ProgPoint` bezieht.

Dieses Argument kann ausgelassen werden. In diesem Fall ist die Position dem Weltkoordinatensystem zugeordnet. Es muss jedoch angegeben werden, ob ein stationärer TCP oder koordinierte externe Achsen verwendet werden.

Die Argumente **Tool** und **\WObj** werden während der Programmierung zur Berechnung des `ProgPoint` und während der Programmabarbeitung zur Berechnung der aktuellen Position verwendet (sofern das Argument `\ExeP` nicht programmiert wurde).

Programmabarbeitung

Die Programmverschiebung bedeutet, dass das `ProgDisp`-Koordinatensystem in Relation zum Objekt-Koordinatensystem verschoben wird. Da alle Positionen auf das `ProgDisp`-Koordinatensystem verweisen, werden auch alle programmierten Positionen verschoben Die Abbildung unten zeigt die parallele Verschiebung einer programmierten Position mithilfe einer Programmverschiebung.



xx0500002186

Fortsetzung auf nächster Seite

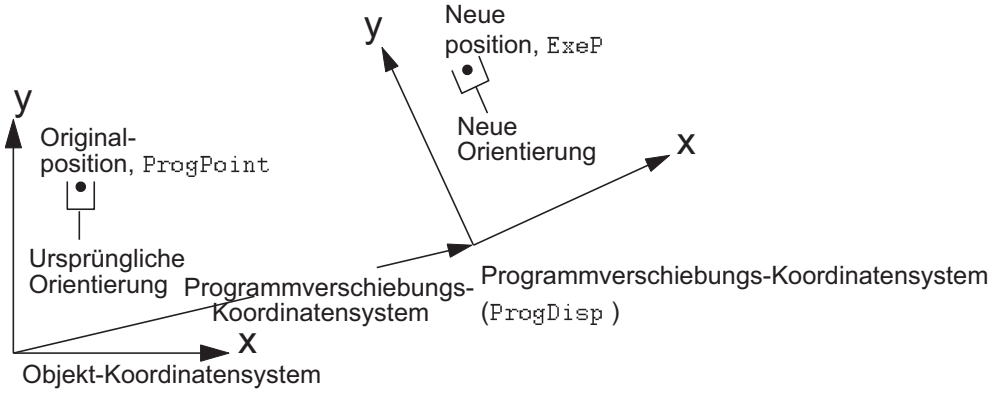
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmverschiebung wird aktiviert, wenn die Instruktion `PDispOn` abgearbeitet wird, und bleibt aktiv, bis eine andere Programmverschiebung aktiviert wird (mit der Instruktion `PDispSet` oder `PDispOn`) oder bis die Programmverschiebung deaktiviert wird (mit der Instruktion `PDispOff`).

Nur eine Programmverschiebung kann jeweils aktiv sein. Andererseits können mehrere `PDispOn`-Instruktionen nacheinander programmiert werden. In diesem Fall werden dann die verschiedenen Programmverschiebungen addiert.

Die Programmverschiebung wird als Differenz zwischen `ExeP` und `ProgPoint` berechnet. Wenn `ExeP` nicht angegeben wurde, wird stattdessen die aktuelle Roboterposition zum Zeitpunkt der Programmabarbeitung verwendet. Deshalb darf sich der Roboter bei der Abarbeitung von `PDispOn` nicht bewegen.

Bei Verwendung des Arguments `\Rot` wird die Rotation auch auf der Basis der Werkzeugorientierung an den beiden Positionen berechnet. Die Verschiebung wird so berechnet, dass die neue Position (`ExeP`) die gleiche Position und Orientierung in Relation zum Programmverschiebungs-Koordinatensystem `ProgDisp` einnimmt wie die alte Position (`ProgPoint`) in Relation zum ursprünglichen Objekt-Koordinatensystem. Die Abbildung unten zeigt die Translation und Rotation einer programmierten Position.



xx0500002187

Die Programmverschiebung wird automatisch zurückgesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `PDispOn` sind unten aufgeführt.

Beispiel 1:

```
PROC draw_square()
  PDispOn *, tool1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

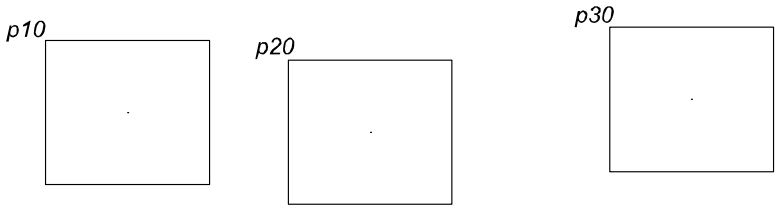
1.169 PDispOn - Aktivieren der Programmverschiebung

RobotWare - OS

Fortsetzung

```
  MoveL *, v500, z10, tool1;
  MoveL *, v500, z10, tool1;
  MoveL *, v500, z10, tool1;
  MoveL *, v500, z10, tool1;
  PDispOff;
ENDPROC
...
MoveL p10, v500, fine \Inpos := inpos50, tool1;
draw_square;
MoveL p20, v500, fine \Inpos := inpos50, tool1;
draw_square;
MoveL p30, v500, fine \Inpos := inpos50, tool1;
draw_square;
```

Die Routine `draw_square` wird verwendet, um auf der Grundlage der Positionen `p10`, `p20` und `p30` dasselbe Bewegungsmuster an drei unterschiedlichen Positionen abzuarbeiten. Die Abbildung unten zeigt, dass durch Programmverschiebungen Bewegungsmuster wiederholt werden können.



xx0500002185

Beispiel 2:

```
SearchL sen1, psearch, p10, v100, tool1\WObj:=fixture1;
PDispOn \ExeP:=psearch, *, tool1 \WObj:=fixture1;
```

Ein Suchvorgang wird ausgeführt, in dem die gesuchte Position des Roboters in der Position `psearch` gespeichert wird. Jede anschließende Bewegung beginnt an dieser Position mithilfe einer Programmverschiebung (parallele Bewegung). Letztere wird anhand der Differenz zwischen der gesuchten Position und dem programmierten Punkt (`*`) berechnet, der in der Instruktion gespeichert wird. Alle Positionen werden im Objekt-Koordinatensystem für die Vorrichtung `fixture1` angegeben.

Syntax

```
PDispOn
  [['\' Rot ]
    ['\' ExeP ':=' <expression (IN) of robtarget>]',']
  [ProgPoint ':='] <expression (IN) of robtarget>','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]';'
```

Weitere Informationen

Information über	Siehe
Deaktivieren der Programmverschiebung	*PDispOff - Deaktivieren der Programmverschiebung auf Seite 536*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Definition einer Programmverschiebung mithilfe eines bekannten Koordinatensystems	*PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem auf Seite 542*
Koordinatensysteme	*Technisches Referenzhandbuch - Systemparameter*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.170 PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem

Verwendung

`PDispSet` (*Program Displacement Set*) wird verwendet, um eine Programmverschiebung in einem bekannten Koordinatensystem zu definieren und zu aktivieren.

Eine Programmverschiebung ist z. B. sinnvoll, wenn ähnliche Bewegungsmuster an mehreren verschiedenen Stellen im Programm verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `PDispSet`:

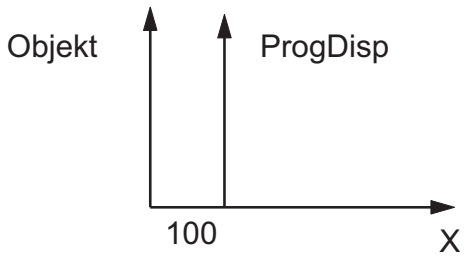
Beispiel 1:

```
VAR pose xp100 := [ [100, 0, 0], [1, 0, 0, 0] ];
...
PDispSet xp100;
```

Aktiviert die Programmverschiebung `xp100`, d. h.:

- Das `ProgDisp`-Koordinatensystem wird vom Objekt-Koordinatensystem um 100 mm in Richtung der positiven x-Achse verschoben (siehe Abbildung unten).
- Solange diese Programmverschiebung aktiv ist, werden alle Positionen um 100 mm in Richtung der x-Achse verschoben.

Die Abbildung zeigt eine 100-mm-Programmverschiebung entlang der x-Achse.



xx0500002199

Argumente

```
PDispSet DispFrame
```

`DispFrame`

Displacement Frame

Datentyp: `pose`

Die Programmverschiebung wird mit dem Datentyp `pose` definiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

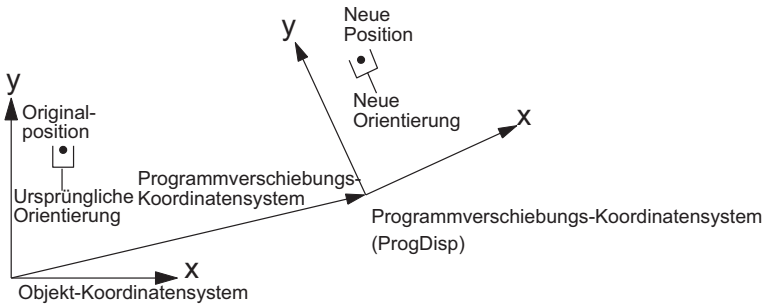
1.170 PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem

RobotWare - OS

Fortsetzung

Programmabarbeitung

Die Programmverschiebung umfasst Translation und/oder Rotation des `ProgDisp`-Koordinatensystems in Relation zum Objekt-Koordinatensystem. Da alle Positionen auf das `ProgDisp`-Koordinatensystem verweisen, werden auch alle programmierten Positionen verschoben. Die Abbildung unten zeigt die Translation und Rotation einer programmierten Position.



xx0500002204

Programmverschiebung wird aktiviert, wenn die Instruktion `PDispOn` abgearbeitet wird, und bleibt aktiv, bis eine andere Programmverschiebung aktiviert wird (mit der Instruktion `PDispSet` oder `PDispOn`) oder bis die Programmverschiebung deaktiviert wird (mit der Instruktion `PDispOff`).

Nur eine Programmverschiebung kann jeweils aktiv sein. Programmverschiebungen können mit `PDispSet` nicht addiert werden.

Die Programmverschiebung wird automatisch zurückgesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Syntax

```
PDispSet
  [ DispFrame ':=' ] < expression (IN) of pose> ';'
```

Weitere Informationen

Information über	Siehe
Deaktivieren der Programmverschiebung	*PDispOff - Deaktivieren der Programmverschiebung auf Seite 536*
Definition einer Programmverschiebung mit zwei Positionen	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Definition von Daten des Typs "pose"	*pose – Koordinatentransformationen auf Seite 1759*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.170 PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem

RobotWare - OS

Fortsetzung

Information über	Siehe
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele zur Verwendung von Programmverschiebungen	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.171 ProcCall - Aufrufen einer neuen Prozedur

Verwendung

Ein Prozeduraufruf wird verwendet, um die Programmabarbeitung an eine andere Prozedur zu versetzen. Nach der vollständigen Abarbeitung der Prozedur fährt die Programmabarbeitung mit der Instruktion fort, die auf den Prozeduraufruf folgt.

Es ist gewöhnlich möglich, eine Reihe von Argumenten an die neue Prozedur zu senden. Diese Argumente steuern das Verhalten der Prozedur und ermöglichen, dass dieselbe Prozedur für unterschiedliche Aufgaben verwendet wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ProcCall`:

Beispiel 1:

```
weldpipe1;
```

Ruft die Prozedur `weldpipe1` auf.

Beispiel 2:

```
errormessage;
Set do1;
...
PROC errormessage()
  TPWrite "ERROR";
ENDPROC
```

Die Prozedur `errormessage` wird aufgerufen. Nach Abschluss dieser Prozedur kehrt die Programmabarbeitung zu der Instruktion zurück, die auf den Prozeduraufruf `Set do1` folgt.

Argumente

```
Procedure { Argument }
```

`Procedure`

Identifier

Der Name der aufzurufenden Prozedur.

`Argument`

Datentyp: In Übereinstimmung mit der Prozedurdeklaration.

Die Argumente der Prozedur (in Einklang mit den Parametern der Prozedur).

Grundlegende Beispiele

Grundlegende Beispiele für die Instruktion `ProcCall` sind unten aufgeführt.

Beispiel 1:

```
weldpipe2 10, lowspeed;
```

Ruft die Prozedur `weldpipe2` mit zwei Argumenten auf.

Beispiel 2:

```
weldpipe3 10 \speed:=20;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.171 ProcCall - Aufrufen einer neuen Prozedur

RobotWare - OS

Fortsetzung

Ruft die Prozedur `weldpipe3` mit einem obligatorischen und einem optionalen Argument auf.

Einschränkungen

Die Argumente der Prozedur müssen zu ihren Parametern passen:

- Alle obligatorischen Argumente müssen angegeben werden.
- Sie müssen sich außerdem in der korrekten Reihenfolge befinden.
- Sie müssen denselben Datentyp aufweisen.
- Sie müssen den korrekten Typ hinsichtlich des Zugriffsmodus aufweisen (Eingang, Variable oder Persistente).

Eine Routine kann eine Routine aufrufen, die wiederum auch eine Routine aufrufen kann. Eine Routine kann auch sich selbst aufrufen, dies wird als rekursiver Aufruf bezeichnet. Die Anzahl der zulässigen Routineebenen hängt von der Anzahl der Parameter ab. Meist sind mehr als 10 Ebenen zulässig.

Syntax

```
<procedure> [ <argument list> ] ';'
```

Weitere Informationen

Information über	Siehe
Argumente, Parameter	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.172 ProcerrRecovery - Generieren und Beheben von Bewegungsfehlern

Verwendung

`ProcerrRecovery` kann verwendet werden, um während einer Roboterbewegung einen Prozessfehler zu generieren und zu behandeln sowie den Prozess und die Bewegung anschließend über eine ERROR-Behandlungsroutine neu zu starten.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ProcerrRecovery`:

Siehe auch *Weitere Beispiele auf Seite 549*.

Die unten angegebenen Beispiele sind nicht realistisch und dienen lediglich zur Verdeutlichung.

Beispiel 1:

```
MoveL p1, v50, z30, tool2;
ProcerrRecovery \SyncOrgMoveInst;
MoveL p2, v50, z30, tool2;
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StartMove;
    RETRY;
  ENDIF
```

Die Roboterbewegung wird auf dem Weg zu `p1` unterbrochen und die Programmabarbeitung wird an die `ERROR`-Behandlungsroutine in der Routine übergeben, von der die ursprüngliche Bahn erstellt wurde, auf der der Fehler aufgetreten ist (in diesem Fall die Bahn zu `MoveL p1`). Die Bewegung wird mithilfe von `StartMove` neu gestartet und die Abarbeitung wird mit `RETRY` fortgesetzt.

Beispiel 2:

```
MoveL p1, v50, fine, tool2;
ProcerrRecovery \SyncLastMoveInst;
MoveL p2, v50, z30, tool2;
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StartMove;
    RETRY;
  ENDIF
```

Die Roboterbewegung zu `p2` wird sofort angehalten. Die Programmabarbeitung wird an die `ERROR`-Behandlungsroutine in der Routine übergeben, in der das Programm beim Auftreten des Fehlers gerade eine Bewegungsinstruktion ausgeführt oder die Ausführung einer Bewegungsinstruktion vorbereitet hat (in diesem Fall `MoveL p2`). Die Bewegung wird mithilfe von `StartMove` neu gestartet und die Abarbeitung wird mit `RETRY` fortgesetzt.

Argumente

```
ProcerrRecovery[\SyncOrgMoveInst] | [\SyncLastMoveInst]
     [\ProcSignal]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.172 ProcerrRecovery - Generieren und Beheben von Bewegungsfehlern
RobotWare - OS
Fortsetzung

[\SyncOrgMoveInst]

Datentyp: switch

Der Fehler kann in der Routine behandelt werden, von der die ursprüngliche Bahn erstellt wurde, auf der der Fehler aufgetreten ist.

[\SyncLastMoveInst]

Datentyp: switch

Der Fehler kann in der Routine behandelt werden, in der das Programm zum Zeitpunkt des Auftretens des Fehlers gerade eine Bewegungsinstruktion ausgeführt hat.

Wenn das Programm zum Zeitpunkt des Fehlers keine Bewegungsinstruktion ausgeführt hat, wird die Übergabe der Ausführung an die ERROR-Behandlungsroutine verschoben, bis das Programm die nächste Bewegungsinstruktion ausführt. Das bedeutet, dass die Übergabe an die ERROR-Behandlungsroutine verschoben wird, wenn sich der Roboter an einem Stopppunkt oder zwischen dem Vorlaufpunkt und der Mitte der Zonenbahn befindet. Der Fehler kann in dieser Routine behandelt werden.

[\ProcSignal]

Datentyp: signaldo

Optionaler Parameter, der dem Benutzer das Ein- und Ausschalten der Instruktionsanwendung ermöglicht. Wenn dieser Parameter verwendet wird und der Signalwert auf 0 steht, wird ein behebbarer Fehler verworfen und kein Prozessfehler erzeugt.

Programmabarbeitung

Das Ausführen von ProcerrRecovery im kontinuierlichen Modus führt zu folgendem Ergebnis:

- Der Roboter wird auf seiner Bahn sofort angehalten.
- Die Variable ERRNO wird auf ERR_PATH_STOP gesetzt.
- Die Ausführung wird gemäß den Vorschriften für asynchron ausgelöste Fehler an eine ERROR-Behandlung übergeben.

Diese Instruktion hat im schrittweisen Modus keinerlei Auswirkungen.

Eine Beschreibung von asynchron aufgetretenen Fehlern, die mit ProcerrRecovery erzeugt wurde, finden Sie unter *Technical reference manual - RAPID kernel*.

ProcerrRecovery kann auch in MultiMove-Systemen verwendet werden, um die Ausführung beim Betrieb im synchronisieren Modus in mehreren Programmtasks an die ERROR-Behandlungsroutine zu übergeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_PATH_STOP	Abarbeitung von ProcerrRecovery im kontinuierlichen Modus.
ERR_PROCSIGNAL_OFF	Der optionale Parameter \ProcSignal wird verwendet und das Signal bei der Abarbeitung der Instruktion ist ausgeschaltet.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion ProcerrRecovery sind unten aufgeführt.

Beispiel mit ProcerrRecovery\SyncOrgMoveInst

```
MODULE user_module
  VAR intnum proc_sup_int;

  PROC main()
    ...
    MoveL p1, v1000, fine, tool1;
    do_process;
    ...
  ENDPROC
  PROC do_process()
    my_proc_on;
    MoveL p2, v200, z10, tool1;
    MoveL p3, v200, fine, tool1;
    my_proc_off;
  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      my_proc_on;
      StartMove;
      RETRY;
    ENDIF
  ENDPROC

  TRAP iprocfail
    my_proc_off;
    ProcerrRecovery \SyncOrgMoveInst;
  ENDTRAP
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.172 ProcerrRecovery - Generieren und Beheben von Bewegungsfehlern

RobotWare - OS

Fortsetzung

```
  PROC my_proc_on()
    SetDO do_myproc, 1;
    CONNECT proc_sup_int WITH iprocfail;
    ISignalDI di_proc_sup, 1, proc_sup_int;
  ENDPROC

  PROC my_proc_off()
    SetDO do_myproc, 0;
    IDelete proc_sup_int;
  ENDPROC
ENDMODULE
```

Asynchron ausgelöste Fehler, die mithilfe von `ProcerrRecovery` und `\SyncOrgMoveInst` generiert wurden, können in diesem Beispiel in der Routine do_process behandelt werden, da die Bahn, auf der der Fehler aufgetreten ist, immer in der Routine `do_process` erstellt wird.

Ein Prozessfluss wird gestartet, indem das Signal `do_myproc` auf `1` gesetzt wird. Das Signal `di_proc_sup` überwacht den Prozess und es wird ein asynchroner Fehler ausgelöst, wenn `di_proc_sup` den Wert `1` annimmt. In diesem einfachen Beispiel wird der Fehler behoben, indem `do_myproc` erneut auf `1` gesetzt wird, bevor die Bewegung fortgesetzt wird.

Example with ProcerrRecovery\SyncLastMoveInst

```
MODULE user_module
  PROC main()
    ...
    MoveL p1, v1000, fine, tool1;
    do_process;
    ...
  ENDPROC
  PROC do_process()
    proc_on;
    proc_move p2, v200, z10, tool1;
    proc_move p3, v200, fine, tool1;
    proc_off;
  ERROR
    IF ERRNO = ERR_PATH_STOP THEN
      StorePath;
      p4 := CRobT(\Tool:=tool1);
      ! Move to service station and fix the problem
      MoveL p4, v200, fine, tool1;
      RestoPath;
      proc_on;
      StartMoveRetry;
    ENDIF
  ENDPROC
ENDMODULE

MODULE proc_module (SYSMODULE, NOSTEPIN)
  VAR intnum proc_sup_int;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  VAR num try_no := 0;

TRAP iprocfail
    proc_off;
    ProcerrRecovery \SyncLastMoveInst;
  ENDTRAP

  PROC proc_on()
    SetDO do_proc, 1;
    CONNECT proc_sup_int WITH iprocfail;
    ISignalDI di_proc_sup, 1, proc_sup_int;
  ENDPROC

  PROC proc_off()
    SetDO do_proc, 0;
    IDelete proc_sup_int;
  ENDPROC

PROC proc_move (robtarget ToPoint, speeddata Speed, zonedata Zone,
     PERS tooldata Tool)
    MoveL ToPoint, Speed, Zone, Tool;
    ERROR
      IF ERRNO = ERR_PATH_STOP THEN
        try_no := try_no + 1;
        IF try_no < 4 THEN
          proc_on;
          StartMoveRetry;
        ELSE
          RaiseToUser \Continue;
      ENDIF
    ENDPROC
ENDMODULE
```

Asynchron ausgelöste Fehler, die mit `ProcerrRecovery` und dem Schalter `\SyncLastMoveInst` generiert wurden, können in diesem Beispiel in der Routine `proc_move` behandelt werden, da alle Bewegungsinstruktionen immer in der Routine proc_move erstellt werden. Wenn sich der Programmzeiger in der Routine `do_process` befindet, wird die Übergabe an die `ERROR`-Behandlungsroutine verzögert, bis in der Routine `proc_move` die nächste Instruktion `MoveL` ausgeführt wird.

Ein Prozessfluss wird gestartet, indem das Signal `do_myproc`auf `1` gesetzt wird. Das Signal `di_proc_sup` überwacht den Prozess und es wird ein asynchroner Fehler ausgelöst, wenn `di_proc_sup` auf `1` gesetzt wird. In diesem einfachen Beispiel wird der Fehler behoben, indem vor dem Fortsetzen der Bewegung `do_myproc` auf `1` gesetzt wird.

Wenn eine vordefinierte `NOSTEPIN`-Routine verwendet wird, empfehlen wir die Verwendung des optionalen switch-Parameters `\SyncLastMoveInst`, da die vordefinierte Routine auf diese Weise entscheiden kann, ob ein Fehler innerhalb der Routine behandelt wird oder vom Endbenutzer behandelt werden muss.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.172 ProcerrRecovery - Generieren und Beheben von Bewegungsfehlern

RobotWare - OS

Fortsetzung

Einschränkungen

Eine Fehlerbehandlung von asynchron ausgelösten Prozessfehlern kann nur erfolgen, wenn der Bewegungstask mit der Bewegungsinstruktion des Prozesses auf der Basisebene ausgeführt wird, wenn der Prozessfehler auftritt. Somit kann keine Fehlerbehandlung erfolgen, wenn der Programmtask mit der Prozessinstruktion in folgenden Komponenten ausgeführt wird:

- alle Ereignisroutinen
- alle Routinenbehandlungen (`ERROR`, `BACKWARD` oder `UNDO`)
- Benutzerausführungsebene (Serviceroutine)

Siehe *Technical reference manual - RAPID kernel*, *Error recovery*, *Asynchronously raised errors*.

Wenn keine Fehlerbehandlungsroutine mit `StartMove` + `RETRY` oder `StartMoveRetry` verwendet wird, bleibt die Programmabarbeitung hängen. Dies kann nur behoben werden, indem der PZ auf main gesetzt wird.

Syntax

```
ProcerrRecovery
  [ '\' SyncOrgMoveInst ] | [' \' SyncLastMoveInst ]
  [ '\' ProcSignal' :=' ] < variable (VAR) of signaldo > ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlungen	*Technisches Referenzhandbuch - RAPID Overview*
Asynchron ausgelöste Fehler	*Technical reference manual - RAPID kernel, Fehlerbehebung*
Weitergeben eines Fehlers an die Benutzerebene	*RaiseToUser - Weitergeben eines Fehlers an die Benutzerebene auf Seite 578*
Wiederaufnehmen der Bewegung und der Programmabarbeitung	*StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung auf Seite 809*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.173 PrxActivAndStoreRecord - Aktiviert und speichert die aufgezeichneten Profildaten

Verwendung

PrxActivAndStoreRecord wird dazu verwendet, die aufgezeichneten Profildaten zu aktivieren und in einer Datei zu speichern.

Kann verwendet werden, statt sowohl `PrxActivRecord` als auch `PrxStoreRecord` aufzurufen.

Grundlegendes Beispiel

```
PrxActivAndStoreRecord SSYNC1, 1, "profile.log";
```

Das Profil der Sensorbewegung wird aktiviert und in der Datei *profile.log* gespeichert.

Argumente

```
PrxActivAndStoreRecord MechUnit Delay File_name
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`Delay`

Datentyp: `num`

Die Verzögerung in Sekunden, mit der die Aufzeichnung zeitlich versetzt werden kann. Der gültige Wertebereich liegt zwischen 0,01 und 0,1. Wird als Wert 0 festgelegt, wird keine Verzögerung hinzugefügt. Die Verzögerung wird nicht im Profil gespeichert, sondern nur zur Aktivierung verwendet. Wenn die Verzögerung zusammen mit einem gespeicherten Profil verwendet wird, muss die Verzögerung erneut mit der Instruktion `PrxUseFileRecord` festgelegt werden.

`File_name`

Datentyp: `string`

Der Name der Datei, in der das Profil gespeichert wird.

Programmabarbeitung

`PrxActivAndStoreRecord` muss mindestens 0,2 Sekunden vor dem Start der Sensorbewegung ausgeführt werden, wenn die Aufzeichnung für die Synchronisierung verwendet werden soll.

Fehlerbehandlung

Die folgenden behebbaren Fehler können erzeugt werden. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_ACTIV_PROF`	Fehler in aktiviertem Profil.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.173 PrxActivAndStoreRecord - Aktiviert und speichert die aufgezeichneten Profildaten

Machine Synchronization

Fortsetzung

Name	Fehlerursache
ERR_STORE_PROF	Fehler in gespeichertem Profil.
ERR_USE_PROF	Fehler in verwendetem Profil.

Syntax

```
PrxActivAndStoreRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Weitere Informationen

Information über	Siehe
Aktiviert die aufgezeichneten Profildaten	*PrxActivRecord - Aktiviert die aufgezeichneten Profildaten auf Seite 555*
Deaktiviert die Aufzeichnung	*PrxDeactRecord - Deaktiviert eine Aufzeichnung auf Seite 558*
Speichert die aufgezeichneten Profildaten	*PrxStoreRecord - Speichert die aufgezeichneten Profildaten auf Seite 568*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.174 PrxActivRecord - Aktiviert die aufgezeichneten Profildaten

Verwendung

PrxActivRecord wird dazu verwendet, ein Profil zu aktivieren, das gerade aufgezeichnet wurde, um es ohne vorheriges Speichern zu verwenden.

Grundlegendes Beispiel

```
PrxActivRecord SSYNC1, 0;
WaitTime 0.2;
SetDO do_startstop_machine, 1;
!Work synchronized with sensor
...
SetDO do_startstop_machine, 0;
```

Die Aufzeichnung des Sensors ist aktiviert und wird für die Vorhersage der Sensorbewegung verwendet, sobald die Aufzeichnung bereit ist.

Argumente

```
PrxActivRecord MechUnit Delay
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`Delay`

Datentyp: `num`

Die Verzögerung in Sekunden, mit der die Aufzeichnung zeitlich versetzt werden kann. Der gültige Wertebereich liegt zwischen 0,01 und 0,1. Wird als Wert 0 festgelegt, wird keine Verzögerung hinzugefügt.

Programmabarbeitung

`PrxActivRecord` muss mindestens 0,2 Sekunden vor dem Start der Bewegung des Förderers ausgeführt werden

Fehlerbehandlung

Die folgenden behebbaren Fehler können erzeugt werden. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_ACTIV_PROF	Fehler in aktiviertem Profil
ERR_STORE_PROF	Fehler in gespeichertem Profil
ERR_USE_PROF	Fehler in verwendetem Profil

Syntax

```
PrxActivRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.174 PrxActivRecord - Aktiviert die aufgezeichneten Profildaten

Machine Synchronization

Fortsetzung

Weitere Informationen

Information über	Siehe
Aktiviert und speichert die aufgezeichneten Profildaten	*PrxActivAndStoreRecord - Aktiviert und speichert die aufgezeichneten Profildaten auf Seite 553*
Deaktiviert die Aufzeichnung	*PrxDeactRecord - Deaktiviert eine Aufzeichnung auf Seite 558*
Speichert die aufgezeichneten Profildaten	*PrxStoreRecord - Speichert die aufgezeichneten Profildaten auf Seite 568*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.175 PrxDbgStoreRecord - Speichert und prüft die aufgezeichneten Profildaten

Verwendung

PrxDbgStoreRecord wird dazu verwendet, eine nicht aktivierte Aufzeichnung für einen Test zu speichern.

Kann dazu verwendet werden, Aufzeichnung zu vergleichen und die Wiederholbarkeit zu prüfen.

Grundlegendes Beispiel

```
PrxDbgStoreRecord SSYNC1, "debug_profile.log";
```

Speichert die Aufzeichnung in der Datei *debug_profile.log*.

Argumente

```
PrxDbgStoreRecord MechUnit Filename
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`File_name`

Datentyp: `string`

Der Name der Datei, in der die Aufzeichnung gespeichert wird.

Syntax

```
PrxDbgStoreRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Weitere Informationen

Information über	Siehe
Aktiviert und speichert die aufgezeichneten Profildaten	*PrxActivAndStoreRecord - Aktiviert und speichert die aufgezeichneten Profildaten auf Seite 553*
Aktiviert die aufgezeichneten Profildaten	*PrxActivRecord - Aktiviert die aufgezeichneten Profildaten auf Seite 555*
Speichert die aufgezeichneten Profildaten	*PrxStoreRecord - Speichert die aufgezeichneten Profildaten auf Seite 568*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.176 PrxDeactRecord - Deaktiviert eine Aufzeichnung

Verwendung

PrxDeactRecord wird dazu verwendet, eine Aufzeichnung zu deaktivieren.

Grundlegendes Beispiel

```
PrxDeactRecord SSYNC1;
```

Die Aufzeichnung des Sensors ist deaktiviert und wird nicht mehr für die Vorhersage der Sensorbewegung verwendet. Die Aufzeichnung kann wieder aktiviert werden.

Argumente

```
PrxDeactRecord MechUnit
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

Einschränkungen

`PrxDeactRecord` sollte während einer Synchronisierung nicht aufgerufen werden.

Syntax

```
PrxDeactRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Weitere Informationen

Information über	Siehe
Aktiviert die aufgezeichneten Profildaten	*PrxActivRecord - Aktiviert die aufgezeichneten Profildaten auf Seite 555*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.177 PrxResetPos - Setzt die Nullposition des Sensors zurück

Verwendung

PrxResetPos wird dazu verwendet, die Nullposition des Sensors zurückzusetzen.

Die Sensorposition wird zur Synchronisierung und die Aufzeichnungsdatei zurückgesetzt, aber der E/A-Signalwert wird beibehalten. Diese Instruktion wird für das Zurücksetzen des Sensoreingangs per Software verwendet, wo kein Sync-Schalter für das Zurücksetzen des E/A-Signals verfügbar ist.

Grundlegendes Beispiel

```
PrxResetPos SSYNC1;
```

Die Sensorposition wird auf Null gesetzt.

Argumente

```
PrxResetPos MechUnit
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

Programmabarbeitung

Die Sensoreinheit muss gestoppt werden (an der gewünschten Nullposition), bevor `PrxResetPos` aufgerufen wird.

Einschränkungen

Darf nicht mit der Karte DSQC 377A verwendet werden.

Diese Instruktion entspricht einem Sync-Schalter. Das Schrittbetrieb-Fenster sollte nach dieser Instruktion als zusätzliche Achsenpositionen 0,0 zeigen.

Syntax

```
PrxResetPos
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Weitere Informationen

Information über	Siehe
Legt eine Referenzposition für den Sensor fest	*PrxSetPosOffset - Legt eine Referenzposition für den Sensor fest auf Seite 561*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.178 PrxResetRecords - Deaktiviert alle Aufzeichnungen und setzt diese zurück

Verwendung

PrxResetRecords wird dazu verwendet, alle Aufzeichnungen zu deaktivieren und zurückzusetzen.

Grundlegendes Beispiel

```
PrxResetRecords SSYNC1;
```

Die Aufzeichnung des Sensors ist deaktiviert und wird nicht mehr für die Vorhersage der Sensorbewegung verwendet und die Aufzeichnungsdaten werden entfernt.

Argumente

```
PrxResetRecords MechUnit
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

Programmabarbeitung

`PrxResetRecords` muss mindestens 0,2 Sekunden vor dem Start der Bewegung des Förderers ausgeführt werden

Syntax

```
PrxResetRecords
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Weitere Informationen

Information über	Siehe
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.179 PrxSetPosOffset - Legt eine Referenzposition für den Sensor fest

Verwendung

`PrxSetPosOffset` wird dazu verwendet, eine Referenzposition für den Sensor festzulegen.

Die Sensorposition wird als Referenz zur Synchronisierung und für die Aufzeichnungsdatei festgelegt. Diese Funktion wird für die Einstellung der Sensorreferenz per Software verwendet, wo kein Sync-Schalter für das Zurücksetzen des E/A-Signals verfügbar ist.

Grundlegendes Beispiel

```
PrxSetPosOffset SSYNC1, reference;
```

Die Sensorposition wird auf den Referenzwert gesetzt.

Argumente

```
PrxSetPosOffset MechUnit Reference
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`Reference`

Datentyp: `num`

Der Referenzwert in Metern (oder Sensoreinheit). Er muss zwischen -5000 und 5000 liegen.

Programmabarbeitung

Die Sensoreinheit muss angehalten werden, bevor `PrxSetPosOffset` aufgerufen wird.

Einschränkungen

Darf nicht mit der Karte DSQC 377A verwendet werden.

Syntax

```
PrxSetPosOffset
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Reference ':=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Setzt die Nullposition des Sensors zurück	*PrxResetPos - Setzt die Nullposition des Sensors zurück auf Seite 559*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.180 PrxSetRecordSampleTime - Legt die Abtastzeit für die Aufzeichnung eines Profils fest

Verwendung

PrxSetRecordSampleTime wird dazu verwendet, die Abtastzeit in Sekunden für die Aufzeichnung eines Profils festzulegen.

Die Standardabtastzeit wird dem Systemparameter *Pos Update time* entnommen, der zum Typ *CAN interface* der Parametergruppe *Process* gehört. Beachten Sie, dass *Pos Update time* die Abtastzeit in Millisekunden angibt, während `PrxSetRecordSampleTime` die Abtastzeit in Sekunden angibt.

Maximal können 300 Stichproben in einem Profil aufgezeichnet werden. Ist eine Aufzeichnung größer als 300 * *Pos Update time*, muss die Abtastzeit erhöht werden.

Grundlegendes Beispiel

Eine Aufzeichnung über 12 Sekunden muss erstellt werden. Die Abtastzeit darf 12/300 = 0,04 nicht unterschreiten. Die Abtastzeit wird daher auf 0,04 Sekunden gestellt.

```
PrxSetRecordSampleTime SSYNC1, 0.04;
```

Argumente

```
PrxSetRecordSampleTime MechUnit SampleTime
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`SampleTime`

Datentyp: `num`

Die Abtastzeit in Sekunden. Die Abtastzeit muss zwischen 0,01 und 0,1 liegen.

Syntax

```
PrxSetRecordSampleTime
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ SampleTime ':=' ] < expression (IN) of num> ';'
```

Weitere Informationen

Information über	Siehe
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.181 PrxSetSyncalarm - Verhalten bei Sync-Alarmen festlegen

Verwendung

PrxSetSyncalarm wird dazu verwendet, das Verhalten von *sync_alarm_signal* gegenüber einem Puls mit spezifischer Dauer festzulegen.

Wird ein Sync-Alarm ausgelöst, pulsiert *Sync_alarm_signal* über den in der Instruktion `PrxSetSyncalarm` festgelegten Zeitraum hinweg. Es kann auch eingestellt werden, dass kein Pulssignal ausgegeben wird, d.h., das Signal bleibt hoch.

Die Standardpulsdauer beträgt 1 Sekunde.

Grundlegende Beispiele

Beispiel 1:

```
PrxSetSyncalarm SSYNC1 \time:=2;
```

Stellt die Pulsdauer von *sync_alarm_signal* auf 2 Sekunden.

Beispiel 2:

```
PrxSetSyncalarm SSYNC1 \NoPulse;
```

Wird der Sync-Alarm ausgelöst, wird *sync_alarm_signal* eingestellt (nicht pulsierend).

Argumente

```
PrxSetSyncalarm MechUnit [\Time] | [\NoPulse]
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`[\Time]`

Datentyp: `num`

Die Pulsdauer in Sekunden. Sie muss zwischen 0,1 und 60 liegen.

Wird `\Time` auf mehr als 60 gestellt, wird kein Pulssignal ausgegeben (entspricht `\NoPulse`).

`[\NoPulse]`

Datentyp: `switch`

Kein Pulssignal wird ausgegeben. Das Signal wird eingestellt, bis eine Instruktion `SupSyncSensorOff` ausgeführt wird:

Syntax

```
PrxSetSyncalarm
  [ MechUnit ':=' ] < expression (IN) of mechunit>
  [ '\' Time ':=' < expression (IN) of num > ]
  | [ '\' NoPulse ] ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.181 PrxSetSyncalarm - Verhalten bei Sync-Alarmen festlegen

Machine Synchronization

Fortsetzung

Weitere Informationen

Information über	Siehe
SupSyncSensorOff - Stopp der synchronisierten Sensorüberwachung	*SupSyncSensorOff - Stopp der synchronisierten Sensorüberwachung auf Seite 854*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.182 PrxStartRecord - Aufzeichnung eines neuen Profils

Verwendung

PrxStartRecord wird dazu verwendet, alle Profildaten zurückzusetzen und ein neues Profil der Sensorbewegung aufzunehmen, sobald *sensor_start_signal* eingestellt wurde.

Damit eine Aufzeichnung durchgeführt werden kann, ist es wichtig, dass die Verbindung zu einem Sensor hergestellt wird (mechanische Einheit deren Geschwindigkeit die Robotergeschwindigkeit beeinflusst). D. h., die Instruktion `WaitSensor` muss vor Beginn der Aufzeichnung ausgeführt werden.

Grundlegendes Beispiel

```
ActUnit SSYNC1;
WaitSensor SSYNC1;
PrxStartRecord SSYNC1, 1, PRX_PROFILE_T1;
WaitTime 0.2;
SetDO do_startstop_machine 1;
```

Das Signal *do_startstop_machine* startet in diesem Beispiel die Sensorbewegung. Das Profil der Sensorbewegung wird aufgezeichnet, sobald die Maschine das Signal *sensor_start_signal* einstellt.

Argumente

```
PrxStartRecord MechUnit, Record_duration, Profile_type
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`Record_duration`

Datentyp: `num`

Bestimmt die Dauer der Aufzeichnung in Sekunden. Der gültige Wertebereich liegt zwischen 0,1 und *Pos Update time* * 300. Wird der Wert 0 verwendet, muss die Instruktion `PrxStopRecord` verwendet werden, um die Aufzeichnung anzuhalten.

`Profile_type`

Datentyp: `num`

Mögliche Werte und ihre Erklärungen werden unten aufgeführt:

Wert	Beschreibung
`PRX_INDEX_PROF`	Die Aufzeichnung wird durch *sensor_start_signal* gestartet.
`PRX_START_ST_PR`	Eine Start- und Stopp-Bewegung kann aufgezeichnet werden. Das *sensor_start_signal* wird verwendet, um die Startbewegung aufzuzeichnen und das *sensor_stop_signal* wird verwendet, um die Stoppbewegung aufzuzeichnen.
`PRX_STOP_ST_PROF`	Das Gleiche gilt für `PRX_START_ST_PR`, wobei sich nur die Reihenfolge der Signale unterscheidet. Zuerst wird das *sensor_stop_signal* verwendet.
`PRX_STOP_M_PROF`	Die Aufzeichnung wird durch *sensor_stop_signal* gestartet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.182 PrxStartRecord - Aufzeichnung eines neuen Profils

Machine Synchronization

Fortsetzung

Wert	Beschreibung
PRX_HPRESS_PROF	Zur Aufzeichnung eines hydraulischen Presswerkzeugs (wo die Sensorposition Null einer offenen Presse entspricht).
PRX_PROFILE_T1	Zur Aufzeichnung von IMM oder einer anderen Maschine (wo die Sensorposition Null einer geschlossenen Presse entspricht).

Programmabarbeitung

`PrxStartRecord` muss mindestens 0,2 Sekunden vor dem Start der Sensorbewegung ausgeführt werden

Syntax

```
PrxStartRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Record_duration ':=' ] < expression (IN) of num > ','
  [ Profile_type ':=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Aufzeichnung eines Profils anhalten	*PrxStopRecord - Aufzeichnung eines Profils anhalten auf Seite 567*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.183 PrxStopRecord - Aufzeichnung eines Profils anhalten

Verwendung

PrxStopRecord wird dazu verwendet, die Aufzeichnung eines Profils anzuhalten.

Sollte immer verwendet werden, wenn für `PrxStartRecord` die `Record_duration` auf 0 gestellt wurde.

Grundlegendes Beispiel

```
ActUnit SSYNC1;
WaitSensor SSYNC1;
PrxStartRecord SSYNC1, 0, PRX_PROFILE_T1;
WaitTime 0.2;
SetDo do_startstop_machine 1;
WaitTime 2;
PrxStopRecord SSYNC1;
```

Das Signal *do_startstop_machine* startet in diesem Beispiel die Sensorbewegung. Das Profil der Sensorbewegung wird aufgezeichnet, sobald *sensor_start_signal* eingestellt wurde und nach zwei Sekunden wird die Aufzeichnung mit der Instruktion `PrxStopRecord` angehalten.

Argumente

```
PrxStopRecord MechUnit
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

Syntax

```
PrxStopRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ';'
```

Weitere Informationen

Information über	Siehe
Aufzeichnung eines neuen Profils	*PrxStartRecord - Aufzeichnung eines neuen Profils auf Seite 565*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.184 PrxStoreRecord - Speichert die aufgezeichneten Profildaten

Verwendung

PrxStoreRecord wird dazu verwendet, die aufgezeichneten Profildaten in einer Datei zu speichern.

Grundlegendes Beispiel

```
ActUnit SSYNC1;
WaitSensor SSYNC1;
PrxStartRecord SSYNC1, 0, PRX_PROFILE_T1;
WaitTime 0.2;
SetDo do_startstop_machine 1;
WaitTime 2;
PrxStopRecord SSYNC1;
PrxActivRecord SSYNC1;
SetDo do_startstop_machine 0;
PrxStoreRecord SSYNC1, 0, "profile.log";
```

Das Profil der Sensorbewegung wird aufgezeichnet, sobald *sensor_start_signal* eingestellt wurde und wird in der Datei *profile.log* gespeichert.

Argumente

```
PrxStoreRecord MechUnit Delay Filename
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`Delay`

Datentyp: `num`

Die Verzögerung in Sekunden, mit der die Aufzeichnung zeitlich versetzt werden kann. Der gültige Wertebereich liegt zwischen 0,01 und 0,1. Wird als Wert 0 festgelegt, wird keine Verzögerung hinzugefügt. Die Verzögerung wird nicht im Profil gespeichert, sondern nur zur Aktivierung verwendet. Wenn die Verzögerung zusammen mit einem gespeicherten Profil verwendet wird, muss die Verzögerung erneut mit der Instruktion `PrxUseFileRecord` festgelegt werden.

`File_name`

Datentyp: `string`

Der Name der Datei, in der das Profil gespeichert wird.

Einschränkungen

Die Aufzeichnung muss aktiviert werden, bevor `PrxStoreRecord` aufgerufen wird.

Syntax

```
PrxStoreRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Aktiviert die aufgezeichneten Profildaten	*PrxActivRecord - Aktiviert die aufgezeichneten Profildaten auf Seite 555*
Deaktiviert die Aufzeichnung	*PrxDeactRecord - Deaktiviert eine Aufzeichnung auf Seite 558*
Verwendung der aufgezeichneten Profildaten	*PrxUseFileRecord - Verwendet die aufgezeichneten Profildaten auf Seite 570*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.185 PrxUseFileRecord - Verwendet die aufgezeichneten Profildaten

Verwendung

PrxUseFileRecord wird dazu verwendet, eine Aufzeichnung aus einer Datei zu laden und zu aktivieren, um einen Sensor zu synchronisieren.

Grundlegendes Beispiel

```
PrxUseFileRecord SSYNC1, 0, "profile.log";
WaitTime 0.2;
SetDo do_startstop_machine 1;
!Work synchronized with sensorWork synchronized with sensor
...
SetDo do_startstop_machine 0;
```

Argumente

```
PrxUseFileRecord MechUnit Delay Filename
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

`Delay`

Datentyp: `num`

Die Verzögerung in Sekunden, mit der die Aufzeichnung zeitlich versetzt werden kann. Der gültige Wertebereich liegt zwischen 0,01 und 0,1. Wird als Wert 0 festgelegt, wird keine Verzögerung hinzugefügt.

`File_name`

Datentyp: `string`

Der Name der Datei, in der das Profil gespeichert wird.

Programmabarbeitung

`PrxUseFileRecord` muss mindestens 0,2 Sekunden vor dem Start der Bewegung des Förderers ausgeführt werden

Syntax

```
PrxUseFileRecord
  [ MechUnit ':=' ] < expression (IN) of mechunit> ','
  [ Delay ':=' ] < expression (IN) of num > ','
  [ File_name ':=' ] < expression (IN) of string > ';'
```

Weitere Informationen

Information über	Siehe
Aktiviert die aufgezeichneten Profildaten	*PrxActivRecord - Aktiviert die aufgezeichneten Profildaten auf Seite 555*
Deaktiviert die Aufzeichnung	*PrxDeactRecord - Deaktiviert eine Aufzeichnung auf Seite 558*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.185 PrxUseFileRecord - Verwendet die aufgezeichneten Profildaten

Machine Synchronization

Fortsetzung

Information über	Siehe
Speichert die aufgezeichneten Profildaten	*PrxStoreRecord - Speichert die aufgezeichneten Profildaten auf Seite 568*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.186 PulseDO - Erzeugen eines Impulses auf einem digitalen Ausgangssignal

Verwendung

`PulseDO` is wird verwendet, um einen Impuls auf einem digitalen Ausgangssignal zu erzeugen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `PulseDO`:

Beispiel 1:

```
PulseDO do15;
```

Ein Impuls mit der Länge 0,2 s wird auf dem Ausgangssignal `do15` erzeugt.

Beispiel 2:

```
PulseDO \PLength:=1.0, ignition;
```

Ein Impuls mit der Länge 1,0 s wird auf dem Signal `ignition` erzeugt.

Beispiel 3

```
! Program task MAIN
PulseDO \High, do3;
! At almost the same time in program task BCK1
PulseDO \High, do3;
```

Ein positiver Impuls (Wert 1) wird von zwei Programmtasks beinahe gleichzeitig auf Signal `do3` erzeugt. Das Resultat ist ein positiver Impuls, der die Standardlänge von 0,2 s um eine Pulslänge überschreitet, d. h. zwei aufeinander folgende Impulse mit jeweils der Länge von 0,2 s.

Argumente

```
PulseDO [ \High ] [ \PLength ] Signal
```

`[ \High ]`

High level

Datentyp: `switch`

Legt fest, dass der ursprüngliche Wert unabhängig vom aktuellen Status immer auf high (1) gesetzt sein muss, wenn die Instruktion abgearbeitet wird.

`[ \PLength ]`

Pulse Length

Datentyp: `num`

Die Länge des Impulses in Sekunden (0,001 bis 2000 s). Wenn das Argument ausgelassen wird, wird ein Impuls mit 0,2 Sekunden erzeugt.

`Signal`

Datentyp: `signaldo`

Der Name des Signals, auf dem ein Impuls generiert werden soll.

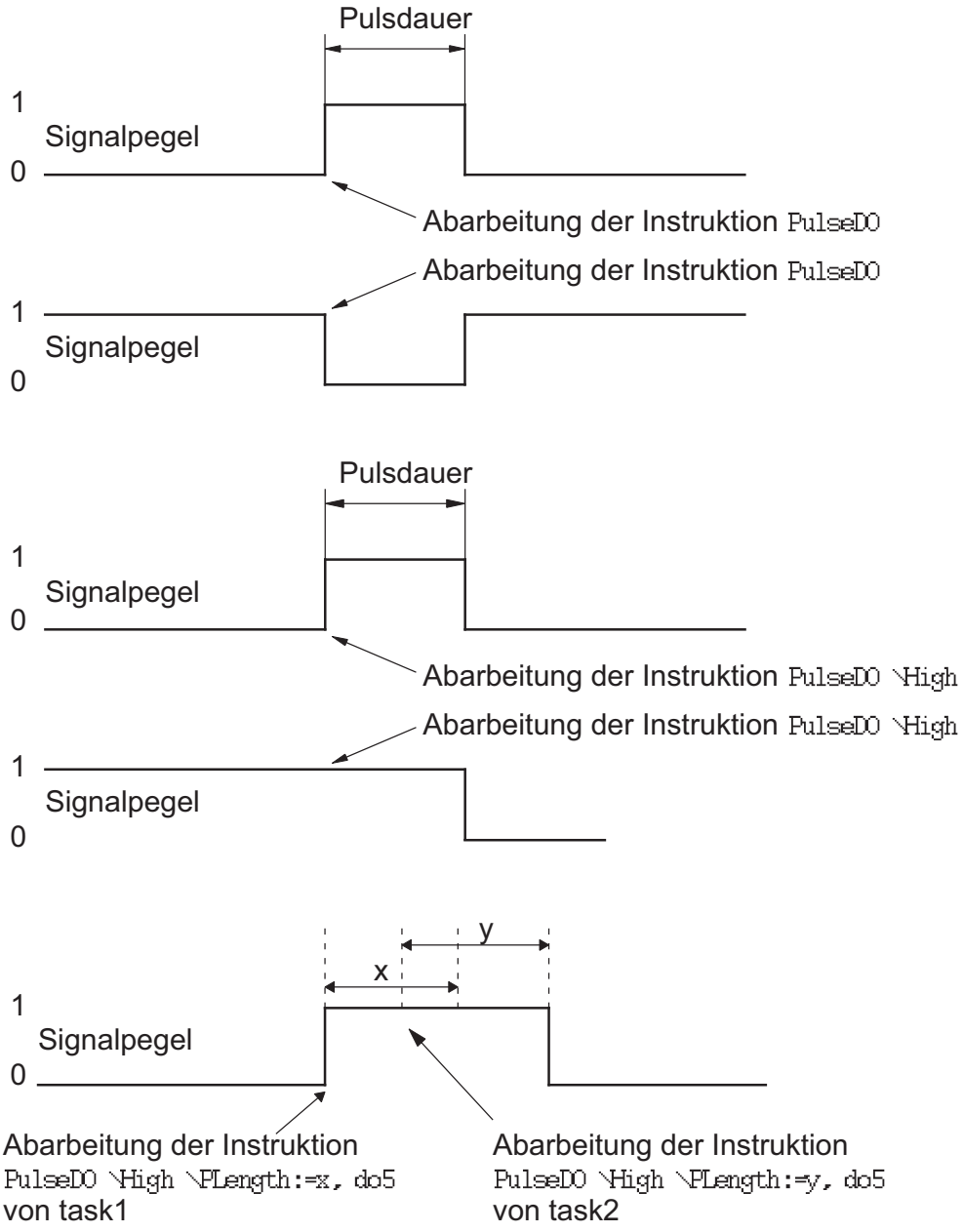
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Die nächste Instruktion nach `PulseDO` wird direkt nach Beginn des Impulses abgearbeitet. Der Impuls kann dann gesetzt/zurückgesetzt werden, ohne die übrige Abarbeitung des Programms zu beeinflussen.

Die Abbildung unten zeigt Beispiele für das Erzeugen eines Impulses auf einem digitalen Ausgangssignal.



xx0500002217

Die nächste Instruktion wird direkt nach Beginn des Impulses abgearbeitet. Der Impuls kann dann gesetzt/zurückgesetzt werden, ohne die übrige Abarbeitung des Programms zu beeinflussen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.186 PulseDO - Erzeugen eines Impulses auf einem digitalen Ausgangssignal

RobotWare - OS

Fortsetzung

Einschränkungen

Die Länge des Impulses hat eine Auflösung von 0,001 Sekunden. Abweichende programmierte Werte werden abgerundet.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
PulseDO
  ['\' High]
  ['\' PLength ':=' <expression (IN) of num>]','
  [Signal ':='] <variable (VAR) of signaldo>';'
```

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.187 RAISE - Ruft die Fehlerbearbeitung auf

Verwendung

RAISE wird verwendet, um einen Fehler innerhalb des Programms zu erzeugen, und dann die Fehlerbehandlung der aktuellen Routine aufzurufen.

RAISE kann auch in der Fehlerbehandlung verwendet werden, um den aktuellen Fehler an die Fehlerbehandlung der aufrufenden Routine zu übergeben.

Mithilfe dieser Instruktion kann beispielsweise ein Rücksprung auf eine höhere Ebene in der Programmstruktur erfolgen, z. B. zur Fehlerbehandlung der Hauptroutine, wenn ein Fehler auf einer niedrigeren Ebene auftritt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion RAISE:

Siehe auch *Weitere Beispiele auf Seite 576*.

Beispiel 1:

```
MODULE MainModule .
VAR errnum ERR_MY_ERR := -1;

PROC main()
  BookErrNo ERR_MY_ERR;

  IF di1 = 0 THEN
    RAISE ERR_MY_ERR;
  ENDIF

ERROR
  IF ERRNO = ERR_MY_ERR THEN
    TPWrite "di1 equals 0";
  ENDIF

ENDPROC

ENDMODULE
```

Wenn in dieser Implementierung di1 gleich 0 ist, wird dies als Fehler betrachtet. RAISE erzwingt die Abarbeitung durch die Fehlerbehandlung. In diesem Beispiel hat der Benutzer eine eigene Fehlernummer erstellt, um den speziellen Fehler zu behandeln.

Argumente

```
RAISE [ Error no. ]
```

Error no.

Datentyp: errnum

Fehlernummer: Eine beliebige Zahl zwischen 1 und 90, anhand deren die Fehlerbehandlungsroutine den aufgetretenen Fehler lokalisieren kann (die Systemvariable ERRNO).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.187 RAISE - Ruft die Fehlerbearbeitung auf
RobotWare-OS
Fortsetzung

Es ist auch möglich, mit der Instruktion BookErrNo. eine Fehlernummer außerhalb des Bereichs 1-90 festzulegen.

Die Fehlernummer muss außerhalb der Fehlerbehandlung in einer RAISE-Instruktion angegeben werden, damit die Abarbeitung an die Fehlerbehandlung dieser Routine übertragen werden kann.

Wenn sich die Instruktion in der Fehlerbehandlung einer Routine befindet, wird der Fehler an die Fehlerbehandlung der aufrufenden Routine weitergeleitet. In diesem Fall muss die Fehlernummer nicht angegeben werden.

Programmabarbeitung

Die Programmabarbeitung fährt in der Fehlerbehandlung der Routine fort. Nach Abarbeiten der Fehlerbehandlung kann die Programmabarbeitung fortfahren mit:

- der Routine, die die betreffende Routine aufgerufen hat (RETURN).
- der Fehlerbehandlung, die die betreffende Routine aufgerufen hat (RAISE).

Eine RAISE-Instruktion in der Fehlerbehandlung einer Routine verfügt noch über eine weitere Funktion. Sie kann für einen langen Sprung verwendet werden (siehe „Fehlerbehandlung mit einem langen Sprung“). Bei einem langen Sprung ist es möglich, einen Fehler von einer Fehlerbehandlung in einem Schritt aus einer tief verschachtelten Aufrufkette an eine höhere Ebene zu verteilen.

Wenn sich die RAISE-Instruktion in einer Trap-Routine befindet, wird der Fehler in der Fehlerbehandlung des Systems behandelt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_ILLRAISE	Fehlernummer in RAISE ist außerhalb des zulässigen Bereichs.

Weitere Beispiele

Weitere Beispiele für die Instruktion RAISE sind unten aufgeführt.

Beispiel 1:

```
MODULE MainModule
VAR num value1 := 10;
VAR num value2 := 0;

PROC main()
  routine1;

ERROR
  IF ERRNO = ERR_DIVZERO THEN
    value2 := 1;
    RETRY;
  ENDIF
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
PROC routine1()
  value1 := 5/value2;!This will lead to an error when value2 is
       equal to 0.
ERROR
  RAISE;
ENDPROC

ENDMODULE
```

In diesem Beispiel führt die Division durch Null zu einem Fehler. In der ERROR-Behandlungsroutine leitet RAISE den Fehler an die ERROR-Behandlungsroutine in der aufrufenden Routine „main“ weiter. Derselbe Fehler bleibt aktiviert. Mit RETRY wird die gesamte Routine „routine1“ erneut ausgeführt.

Syntax

```
RAISE [<error number>] ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlung	*Technisches Referenzhandbuch - Systemparameter*
Fehlerbehandlung mit langem Sprung	*Technical reference manual - RAPID kernel*
Vergeben von Fehlernummern	*BookErrNo - Vergeben einer RAPID-Systemfehlernummer auf Seite 50*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.188 RaiseToUser - Weitergeben eines Fehlers an die Benutzerebene

Verwendung

`RaiseToUser` wird in Fehlerbehandlungen in NOSTEPIN-Routinen verwendet, um den aktuellen Fehler oder einen beliebigen anderen definierten Fehler an die Fehlerbehandlung auf Benutzerebene weiterzuleiten. Die Benutzerebene ist in diesem Fall die erste Routine in einer Aufruffolge über einer NOSTEPIN-Routine.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RaiseToUser`:

Beispiel 1:

```
MODULE MyModule
  VAR errnum ERR_MYDIVZERO:= -1;
  PROC main()
      BookErrNo ERR_MYDIVZERO;
      ...
      routine1;
      ...
      ERROR
      IF ERRNO = ERR_MYDIVZERO THEN
        TRYNEXT;
      ELSE
        RETRY;
      ENDIF
  ENDPROC
ENDMODULE
MODULE MySysModule (SYSMODULE, NOSTEPIN, VIEWONLY)
  PROC routine1()
    ...
    routine2;
    ...
  UNDO
  ! Free allocated resources
  ENDPROC
  PROC routine2()
    VAR num n:=0;
    ...
    reg1:=reg2/n;
    ...
    ERROR
    IF ERRNO = ERR_DIVZERO THEN
      RaiseToUser \Continue \ErrorNumber:=ERR_MYDIVZERO;
    ELSE
      RaiseToUser \BreakOff;
    ENDIF
  ENDPROC
ENDMODULE
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Division durch null in `routine2` wird an die Fehlerbehandlungsroutine in der Routine „main“ weitergeleitet und ERRNO wird auf `ERR_MYDIVZERO` gesetzt. Die Instruktion `TRYNEXT` in der main-Fehlerbehandlungsroutine sorgt dann dafür, dass die Programmabarbeitung mit der Instruktion nach der Division durch null in zero in `routine2` fortgesetzt wird. Dieses Verhalten wird über „\Continue“ gesteuert.

Falls in `routine2` ein weiterer Fehler auftritt, wird durch \BreakOff die Fortsetzung der Ausführung ab der Fehlerbehandlung in der Hauptroutine erzwungen. In diesem Fall wird in routine1 die Abbruchbehandlung ausgeführt, während der Fehler an die Benutzerebene gemeldet wird. Mit der Instruktion RETRY in der Fehlerbehandlung der Hauptroutine wird routine1 von Anfang an neu ausgeführt.

Die Abbruchbehandlung in "routine1" wird auch im Falle von "\Continue" ausgeführt, wenn auf Benutzerebene eine der Instruktionen `RAISE` oder `RETURN` ausgeführt wird.

Argumente

```
RaiseToUser[ \Continue] | [ \BreakOff][ \ErrorNumber]
```

`[\Continue]`

Datentyp: `switch`

Setzt die Abarbeitung in der Routine fort, von der der Fehler ausgelöst wurde.

`[\BreakOff]`

Datentyp:`switch`

Bricht die Aufruffolge ab und setzt die Abarbeitung auf Benutzerebene fort. Sämtliche Abbruchbehandlungen in der Aufruffolge mit Ausnahme derjenigen, die den Fehler ausgelöst hat, werden ausgeführt.

Eines der Argumente `\Continue` oder `\BreakOff` muss programmiert werden, um einen Abarbeitungsfehler zu vermeiden.

`[\ErrorNumber]`

Datentyp: `errnum`

Eine beliebige Zahl zwischen 1 und 90, anhand deren die Fehlerbehandlungsroutine den aufgetretenen Fehler lokalisieren kann (die Systemvariable `ERRNO`).

Es ist auch möglich, mit der Instruktion mit der Instruktion `BookErrNo` eine Fehlernummer außerhalb des Bereichs 1-90 festzulegen.

Falls das Argument `\ErrorNumber` nicht angegeben ist, wird die ursprüngliche Fehlernummer an die Fehlerbehandlung in der Routine auf Benutzerebene weitergeleitet.

Programmabarbeitung

`RaiseToUser` can kann nur in einer Fehlerbehandlung in einer nostepin-Routine verwendet werden.

Die Programmabarbeitung fährt in der Fehlerbehandlung der Routine auf Benutzerebene fort. Die Fehlernummer bleibt aktiv, wenn der optionale Parameter `\ErrorNumber` nicht vorhanden ist. Der Fehler wird von der Fehlerbehandlung des Systems behandelt, wenn auf Benutzerebene keine Fehlerbehandlung

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.188 RaiseToUser - Weitergeben eines Fehlers an die Benutzerebene

RobotWare - OS

Fortsetzung

vorhanden ist. Wenn keines der beiden Argumente `\Continue` oder `\BreakOff` angegeben ist, wird die Fehlerbehandlung des Systems aufgerufen.

Nach der Ausführung der Fehlerbehandlung sind zwei verschiedene Verhaltensweisen möglich. Die Programmabarbeitung fährt in der Routine mit `RaiseToUser` fort, wenn `\Continue` aktiviert ist. Die Programmabarbeitung wird auf Benutzerebene fortgesetzt, wenn `\BreakOff` aktiviert ist.

Die Programmabarbeitung kann folgendermaßen fortgesetzt werden:

- mit der Instruktion, die den Fehler verursacht hat (`RETRY`)
- mit der folgenden Instruktion (`TRYNEXT`)
- mit der Fehlerbehandlung, die die Routine auf Benutzerebene aufgerufen hat (`RAISE`)
- mit der Routine, die die Routine auf Benutzerebene aufgerufen hat (`RETURN`)

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ILLRAISE`	Fehlernummer in `RAISE` ist außerhalb des zulässigen Bereichs.

Syntax

```
RaiseToUser
  [ '\' Continue ]
  '|' [ '\' BreakOff ]
  [ '\' ErrorNumber ':=' ] < expression (IN) of errnum> ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview*
Abbruchbehandlung	*Technisches Referenzhandbuch - RAPID Overview*
Vergeben von Fehlernummern	*BookErrNo - Vergeben einer RAPID-Systemfehlernummer auf Seite 50*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.189 ReadAnyBin – Lesen von Daten aus einem binären E/A-Gerät oder einer Binärdatei

Verwendung

`ReadAnyBin` (*Read Any Binary*) wird verwendet, um alle Datentypen aus einem E/A-Gerät oder einer Binärdatei zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ReadAnyBin`:

Beispiel 1:

```
VAR iodev file1;
VAR robtarget next_target;
...
Open "HOME:" \File:= "bin_file.txt", file1 \Read \Bin;
ReadAnyBin file1, next_target;
```

Das nächste abzuarbeitende Roboterziel `next_target` wird von der Datei `file1` abgelesen.

Argumente

```
ReadAnyBin IODevice Data [\Time]
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) des binären E/A-Geräts oder der Binärdatei, die gelesen werden soll.

`Data`

Datentyp: `anytype`

Die `VAR` bzw. `PERS`, in der die gelesenen Daten gespeichert werden.

`[\Time]`

Datentyp: `num`

Die maximale Dauer für den Lesevorgang (Timeout) in Sekunden. Wenn dieses Argument nicht angegeben wird, beträgt die maximale Dauer 60 Sekunden. Verwenden Sie für eine unendliche Wartezeit die vordefinierte Konstante `WAIT_MAX`.

Wenn diese Zeit abläuft, bevor der Lesevorgang abgeschlossen ist, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_DEV_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Die Timeout-Funktion ist auch während des Programmstopps in Gebrauch und wird vom RAPID-Programm beim Programmstart benachrichtigt.

Programmabarbeitung

Die erforderliche Anzahl an Bytes wird für die angegebenen Daten vom angegebenen binären E/A-Gerät oder aus der binären Datei gelesen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.189 ReadAnyBin – Lesen von Daten aus einem binären E/A-Gerät oder einer Binärdatei

RobotWare - OS

Fortsetzung

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Der Pfad zeigt auf ein nicht vorhandenes Verzeichnis oder es sind zu viele Verzeichnisse gleichzeitig geöffnet.
`ERR_DEV_MAXTIME`	Timeout beim Abarbeiten einer `ReadBin`-, `ReadNum`-, `ReadStr`-, `ReadStrBinReadAnyBin`- oder `ReadRawBytes`-Instruktion.
`ERR_RANYBIN_CHK`	Prüfsummenfehler bei Datenübertragung mit der Instruktion `ReadAnyBin`.
`ERR_RANYBIN_EOF`	Dateiende erkannt, bevor alle Bytes in der Instruktion `ReadAnyBin` gelesen wurden.

Einschränkungen

Diese Instruktion kann nur für E/A-Geräte oder Dateien verwendet werden, die zum binären Lesen geöffnet wurden.

Die mit der Instruktion `ReadAnyBin` zu lesenden Daten müssen von einem Wertdatentyp sein, z. B. `num`, `bool` oder `string`. Ein Datensatz, eine Datensatzkomponente, ein Datenfeld oder eine Datenfeldkomponente dieser Wertdatentypen können ebenfalls verwendet werden. Vollständige Daten oder Teildaten mit einem festen Wert (Semi-Value) oder keinem Wert (Non-Value) können nicht verwendet werden.

Hinweis

Die Variable `VAR` oder `PERS` für die Speicherung der gelesenen Daten kann in mehreren Schritten aktualisiert werden. Warten Sie daher immer, bis die gesamte Datenstruktur aktualisiert ist, bevor Sie von einer `TRAP`-Routine oder einer anderen Programmtask aus Daten lesen.

Weil `WriteAnyBin-ReadAnyBin` nur interne binäre Steuerungsdaten mit einem E/A-Gerät oder mit Dateien zwischen oder in der Robotersteuerung behandelt, wird kein Datenprotokoll veröffentlicht, und die Daten können auf keinem PC interpretiert werden.

Die Entwicklung von Steuerungssoftware kann dazu führen, dass die Kompatibilität nicht mehr gegeben ist. Daher ist es nicht möglich, `WriteAnyBin-ReadAnyBin` in unterschiedlichen Versionen von RobotWare zu verwenden

Syntax

```
ReadAnyBin
  [IODevice ':='] <variable (VAR) of iodev>','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.189 ReadAnyBin – Lesen von Daten aus einem binären E/A-Gerät oder einer Binärdatei


```
[Data ':='] <var or pers (INOUT) of anytype>
['\' Time ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Öffnen von E/A-Geräten oder Dateien	*Technisches Referenzhandbuch - RAPID Overview*
Schreiben von Daten auf binäre/s E/A-Gerät oder Datei	*WriteAnyBin - Schreibt Daten in eine zeichenorientierte Datei oder ein zeichenorientiertes E/A-Gerät auf Seite 1117*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.190 ReadBlock - Lesen eines Datenblocks von einem Gerät

Verwendung

`ReadBlock` wird verwendet, um einen Datenblock von einem Gerät zu lesen, das mit der seriellen Sensorschnittstelle verbunden ist.*. The data is stored in a file.*

Die Sensorschnittstelle kommuniziert mithilfe des RTP1-Transportprotokolls mit zwei Sensoren über serielle Kanäle.

Dies ist ein Beispiel für eine Sensorkanalkonfiguration.

COM_PHY_CHANNEL:

- Name "COM1:"
- Connector "COM1"
- Baudrate 19200

COM_TRP:

- Name "sen1:"
- Type "RTP1"
- PhyChannel "COM1"

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ReadBlock`:

Beispiel 1:

```
CONST string SensorPar := "flp1:senpar.cfg";
CONST num ParBlock:= 1;

! Connect to the sensor device "sen1:" (defined in sio.cfg).
SenDevice "sen1:";

! Read sensor parameters from sensor datablock 1
! and store on flp1:senpar.cfg

ReadBlock "sen1:", ParBlock, SensorPar;
```

Argumente

```
ReadBlock device BlockNo FileName [ \TaskName ]
```

`device`

Datentyp: string

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`BlockNo`

Datentyp: `num`

Das Argument `BlockNo` wird verwendet, um den Datenblock im Sensor auszuwählen, der ausgelesen werden soll.

`FileName`

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das Argument `FileName` wird verwendet, um eine Datei zu definieren, in der die Daten des mithilfe des Arguments `BlockNo` ausgewählten Sensordatenblocks gespeichert werden sollen.

`[ \TaskName ]`

Datentyp: `string`

Das Argument `TaskName` ermöglicht, auf Geräte in anderen RAPID-Tasks zuzugreifen.

Fehlerbehandlung

Fehlerkonstante (ERRNO Wert)	Beschreibung
SEN_NO_MEAS	Messfehler
SEN_NOREADY	Sensor kann Befehl nicht handhaben
SEN_GENERRO	Allgemeiner Sensorfehler
SEN_BUSY	Sensor belegt
SEN_UNKNOWN	Unbekannter Sensor
SEN_EXALARM	Externer Sensorfehler
SEN_CAALARM	Interner Sensorfehler
SEN_TEMP	Sensortemperaturfehler
SEN_VALUE	Unzulässiger Kommunikationswert
SEN_CAMCHECK	Sensorprüffehler
SEN_TIMEOUT	Kommunikationsfehler

Syntax

```
ReadBlock
  [ device ':=' ] < expression(IN) of string> ','
  [ BlockNo ':=' ] < expression (IN) of num > ','
  [ FileName ':=' ] < expression (IN) of string > ','
  [ '\' TaskName ':=' < expression (IN) of string > ] ';'
```

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Schreiben einer Sensorvariablen	*WriteVar – Variable schreiben auf Seite 1132*
Lesen einer Sensorvariablen	*ReadVar - Lesen einer Variablen von einem Gerät auf Seite 1474*
Schreiben eines Sensordatenblocks	*WriteBlock – Datenblock auf Gerät schreiben auf Seite 1121*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.191 ReadCfgData - Lesen von Attributen eines Systemparameters

Verwendung

`ReadCfgData` wird verwendet, um ein Attribut eines benannten Systemparameters (Konfigurationsdaten) zu lesen.

Außer dem Lesen von benannten Parametern können auch nicht benannte Parameter gesucht werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ReadCfgData`. In beiden Beispielen wird gezeigt, wie benannte Parameter gelesen werden.

Beispiel 1:

```
VAR num offset1;
...
ReadCfgData "/MOC/MOTOR_CALIB/rob1_1","cal_offset",offset1;
```

Liest den Kalibrieroffsetwert für axis 1 für `rob_1` in die `num`-Variable `offset1`.

Beispiel 2:

```
VAR string io_device;
...
ReadCfgData "/EIO/EIO_SIGNAL/process_error","Device",io_device;
```

Liest den Namen der E/A-Einheit, in der das Signal `process_error` definiert ist, in die `string`-Variable `io_device`.

Argumente

```
ReadCfgData InstancePath Attribute CfgData [\ListNo]
```

`InstancePath`

Datentyp: `string`

Gibt den Pfad zu dem Parameter an, auf den zugegriffen werden soll.

Für benannte Parameter ist das Format dieser Zeichenfolge `/DOMAIN/TYPE/ParameterName`.

Für unbenannte Parameter ist das Format dieser Zeichenfolge `/DOMAIN/TYPE/Attribute/AttributeValue`.

`Attribute`

Datentyp: `string`

Der Name des Attributs des zu lesenden Parameters.

`CfgData`

Datentyp: `anytype`

Die Variable, in der der Attributwert gespeichert wird. Je nach Attributtyp sind folgende Typen zulässig: `bool`, `num`, `dnum` oder `string`.

`[\ListNo]`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Variable, in der die Instanznummer des zu suchenden und zu aktualisierenden `Attribute` + `AttributeValue` gespeichert wird.

Das erste Vorkommen von `Attribute` + `AttributeValue` hat die Instanznummer 0. Wenn mehrere Instanzen gesucht werden, wird der in `\ListNo` zurückgegebene Wert um 1 erhöht. Wenn hingegen keine weiteren Instanzen vorhanden sind, ist der zurückgegebene Wert -1. Mit der vordefinierten Konstanten `END_OF_LIST` kann überprüft werden, ob mehrere zu suchende Instanzen vorhanden sind.

Programmabarbeitung

Der durch das Argument `Attribute` angegebene Attributwert wird in der durch das Argument `CfgData` angegebenen Variable gespeichert.

Bei Verwendung des Formats `/DOMAIN/TYPE/ParameterName` in `InstancePath` kann nur auf benannte Parameter zugegriffen werden, d. h. Parameter, deren erstes Attribut `name`, `Name` oder `NAME` lautet.

Verwenden Sie für unbenannte Parameter den optionalen Parameter `\ListNo`, um auszuwählen, aus welcher Instanz der Attributwert gelesen werden soll. Er wird nach jedem erfolgreichen Lesen mit der nächsten verfügbaren Instanz aktualisiert.

Vordefinierte Daten

Mithilfe der vordefinierten Konstanten `END_OF_LIST` mit dem Wert -1 kann der Lesevorgang beendet werden, wenn keine weiteren Instanzen gefunden werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_CFG_NOTFND`	In der Konfigurationsdatenbank konnten die Daten mit der Kennzeichnung “`InstancePath` + `Attribute`” nicht gefunden werden.
`ERR_CFG_ILLTYPE`	Der Datentyp für den Parameter `CfgData` ist nicht mit dem echten Datentyp für die gefundenen Daten identisch, die in der Konfigurationsdatenbank mit “`InstancePath` + `Attribute`” angegeben werden.
`ERR_CFG_INTERNAL`	Interner Parameter darf nicht gelesen werden
`ERR_CFG_OUTOFBOUNDS`	Der Wert der Variablen im Argument `\ListNo` liegt bei der Abarbeitung der Instruktion außerhalb des Bereichs verfügbarer Instanzen (0 ... n).

Weitere Beispiele

Weitere Beispiele für die Instruktion `ReadCfgdata` sind unten aufgeführt. In beiden Beispielen wird gezeigt, wie unbenannte Parameter gelesen werden.

Beispiel 1:

```
VAR num list_index;
VAR string read_str;
...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.191 ReadCfgData - Lesen von Attributen eines Systemparameters

RobotWare - OS

Fortsetzung

```
list_index:=0;
ReadCfgData "/EIO/EIO_CROSS/Act1/do_13", "Res", read_str,
     \ListNo:=list_index;
IF read_str <> "" THEN
  TPWrite "Resultant signal for signal do_13 is: " + read_str;
ENDIF
```

Liest das resultierende Signal für das unbenannte digitale Aktorensignal `di_13` und fügt den Namen in die `string`-Variable `read_str` ein.

In diesem Beispiel weist die Domäne EIO folgenden Konfigurationscode auf:

EIO_CROSS:

-Name "Cross_di_1_do_2" -Res "di_1" -Act1 "do_2"

-Name "Cross_di_2_do_2" -Res "di_2" -Act1 "do_2"

-Name "Cross_di_13_do_13" -Res "di_13" -Act1 "do_13"

Beispiel 2:

```
VAR num list_index;
VAR string read_str;
...
list_index:=0;
WHILE list_index <> END_OF_LIST DO
  read_str:="";
  ReadCfgData "/EIO/EIO_SIGNAL/Device/USERIO", "Name", read_str,
       \ListNo:=list_index;
  IF read_str <> "" THEN
    TPWrite "Signal: " + read_str;
  ENDIF
ENDWHILE
..
ERROR
  TRYNEXT;
```

Liest die Namen aller für die E/A-Einheit `USERIO` definierten Signale.

In diesem Beispiel weist die Domäne EIO folgenden Konfigurationscode auf:

```
EIO_SIGNAL:
  -Name "USERDO1" -SignalType "DO" -Device "USERIO" -DeviceMap "0"
  -Name "USERDO2" -SignalType "DO" -Device "USERIO" -DeviceMap "1"
  -Name "USERDO3" -SignalType "DO" -Device "USERIO" -DeviceMap "2"
```

Beispiel 3

```
VAR num list_index;
VAR string read_str;
...
list_index:=0;
WHILE list_index <> END_OF_LIST DO
  read_str:="";
  ReadCfgData "/EIO/DEVICENET_DEVICE/Network/DeviceNet", "Name",
       read_str, \ListNo:=list_index;
  IF read_str <> "" THEN
    TPWrite read_str;
  ENDIF
ENDWHILE
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
..
ERROR
  TRYNEXT;
```

Liest die Namen aller DeviceNet-Geräte.

In diesem Beispiel weist die Domäne EIO folgenden Konfigurationscode auf:

```
DEVICENET_DEVICE:
  -Name PANEL -Network "DeviceNet" -Simulated
  -Name DRV_1 -Network "DeviceNet" -Simulated
  -Name DEVICE1 -Network "DeviceNet" -Simulated
  -Name DEVICE2 -Network "DeviceNet" -Simulated
```

Einschränkungen

Die Umrechnung der Systemparametereinheiten (m, Radiant, Sekunde usw.) in RAPID-Programmeinheiten (mm, Grad, Sekunde usw.) für `CfgData` der Datentypen `num` und `dnum` muss vom Benutzer im RAPID-Programm durchgeführt werden.

Bei Verwendung des Formats `/DOMAIN/TYPE/ParameterName` in `InstancePath` kann nur auf benannte Parameter zugegriffen werden, d. h. Parameter, deren erstes Attribut `name`, `Name` oder `NAME` lautet.

Die Anzahl der Zeichen in RAPID-Zeichenfolgen ist auf 80 beschränkt. Theoretisch könnte das für die Definition von `InstancePath`, `Attribute` oder `CfgData` in einigen Fällen nicht ausreichen.

Syntax

```
ReadCfgData
  [ InstancePath ':=' ] < expression (IN) of string > ','
  [ Attribute ':=' ] < expression (IN) of string > ','
  [ CfgData ':=' ] < variable (VAR) of anytype >
  [ '\' ListNo ':=' < variable (VAR) of num >] ';'
```

Weitere Informationen

Information über	Siehe
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Schreibattribut eines Systemparameters	*WriteCfgData - Schreibt ein Attribut eines Systemparameters auf Seite 1123*
Abrufen des Namens der aktuellen Task	*RobName - Abrufen des TCP-Roboternamens auf Seite 1481*
Konfiguration	*Technisches Referenzhandbuch - Systemparameter*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.192 ReadErrData - Abrufen von Informationen über einen Fehler

Verwendung

`ReadErrData` wird in einer Interrupt-Routine verwendet, um Informationen (Domäne, Typ, Nummer und eingebettete Zeichenfolgen %s) über einen Fehler, eine Statusänderung oder eine Warnung zu beziehen, die die Abarbeitung der Interrupt-Routine verursacht hat.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ReadErrData`:

Siehe Kapitel *Weitere Beispiele auf Seite 591*

Beispiel 1:

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
VAR string titlestr;
VAR string string1;
VAR string string2;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number,
  err_type \Title:=titlestr \Str1:=string1 \Str2:=string2;
ENDTRAP
```

Wenn ein Fehler in der Interrupt-Routine `trap_err` abgefangen wird, werden Fehlerdomäne, Fehlernummer, Fehlertyp und die beiden ersten eingebetteten Zeichenfolgen in der Fehlermeldung in entsprechenden Variablen gespeichert.

Argumente

```
ReadErrData TrapEvent ErrorDomain ErrorId ErrorType [\Title] [\Str1]
     [\Str2] [\Str3] [\Str4] [\Str5]
```

`TrapEvent`

Datentyp: `trapdata`

Variable, die die Informationen über die Ursache des Interrupts enthält.

`ErrorDomain`

Datentyp: `errdomain`

Die Variable zur Speicherung der Fehlerdomäne, der der aufgetretene Fehler, die Statusänderung oder die Warnung angehört (siehe auch den vordefinierten Datentyp `errdomain`).

`ErrorId`

Datentyp: `num`

Die Variable zur Speicherung der Nummer des aufgetretenen Fehlers. Die Fehlernummer wird ohne die erste Ziffer (Fehlerdomäne) und ohne die führenden Nullen der vollständigen Fehlernummer zurückgegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel: 10008 Programm-Neustart, wird als 8 zurückgegeben.

`ErrorType`

Datentyp: `errtype`

Die Variable zur Speicherung des Ereignistyps wie beispielsweise des aufgetretenen Fehlers, der Statusänderung oder der Warnung (siehe auch den vordefinierten Datentyp `errtype`).

`[ \Title ]`

Datentyp: `string`

Variable zum Speichern des Titels in der Fehlermeldung. Der Titel ist im Format UTF8 und es werden nicht für alle Sprachen auf dem FlexPendant alle Zeichen korrekt angezeigt.

`[ \Str1 ] ... [ \Str5 ]`

Datentyp: `string`

Aktualisiert die angegebene Zeichenfolgenvariable mit einem Argument, das in die Fehlermeldung eingebettet ist. In einer Meldung vom Typ %s, %f, %d oder %ld können bis zu fünf Argumente vorhanden sein, die bei der Ausführung dieser Instruktion immer in eine Zeichenfolge konvertiert werden. `Str1` enthält das erste Argument, `Str2` enthält das zweite Argument usw. Informationen über die Anzahl der Argumente in einer Meldung finden Sie in der *Bedienanleitung - Fehlerbehebung, IRC5* . Die eingebetteten Argumente werden in diesem Dokument als „arg“ gekennzeichnet.

Programmabarbeitung

Die Variablen `ErrorDomain`, `ErrorId`, `ErrorType`, `Title` und `Str1 ... Str5` werden gemäß dem Inhalt von `TrapEvent` aktualisiert.

Wenn verschiedene Ereignisse mit derselben Trap-Routine verbunden sind, muss das Programm sicherstellen, dass sich das Ereignis auf die Fehlerüberwachung bezieht. Sie erreichen dies durch den Test, ob `INTNO` mit der Interrupt-Nummer übereinstimmt, die in der Instruktion `IError` verwendet wurde.

Weitere Beispiele

Weitere Beispiele für die Instruktion `ReadErrData` sind unten aufgeführt.

Beispiel 1:

```
VAR intnum err_interrupt;
VAR trapdata err_data;
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
...
PROC main()
  CONNECT err_interrupt WITH trap_err;
  IError COMMON_ERR, TYPE_ERR, err_interrupt;
  ...
  IDelete err_interrupt;
  ...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.192 ReadErrData - Abrufen von Informationen über einen Fehler

RobotWare - OS

Fortsetzung

```
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
  ! Set domain no 1 ... 11
  SetGO go_err1, err_domain;
  ! Set error no 1 ...9999
  SetGO go_err2, err_number;
ENDTRAP
```

Beim Auftreten eines Fehlers (nur Fehler, nicht Warnung oder Statusänderung) wird die Fehlernummer in der Interrupt-Routine abgerufen und mit ihrem Wert werden zwei Gruppen von digitalen Ausgangssignalen gesetzt.

Einschränkung

Es ist nicht möglich, Informationen über interne Fehler anzufordern.

Syntax

```
ReadErrData
  [TrapEvent ':='] <variable (VAR) of trapdata>','
  [ErrorDomain' :='] <variable (VAR) of errdomain>','
  [ErrorId':='] <variable (VAR) of num>','
  [ErrorType' :='] <variable (VAR) of errtype>
  ['\'Title ':='<variable (VAR) of string>]
  ['\'Str1 ':='<variable (VAR) of string>]
  ['\'Str2 ':='<variable (VAR) of string>]
  ['\'Str3 ':='<variable (VAR) of string>]
  ['\'Str4 ':='<variable (VAR) of string>]
  ['\'Str5 ':='<variable (VAR) of string>]';'
```

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Fehlerdomänen, vordefinierte Konstanten	*errdomain – Fehlerdomäne auf Seite 1692*
Fehlertypen, vordefinierte Konstanten	*errtype – Fehlertyp auf Seite 1703*
Anfordern eines Interrupts bei Fehlern	*IError - Anfordern eines Interrupts bei Fehlern auf Seite 255*
Abrufen von Interrupt-Daten für aktuellen Interrupt	*GetTrapData - Abrufen von Interrupt-Daten für aktuellen Interrupt auf Seite 238*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.193 ReadRawBytes - Lesen der rawbytes-Daten

Verwendung

`ReadRawBytes` dient zum Lesen von Daten vom Typ `rawbytes` von einem Gerät, das mit `Open \Bin` geöffnet wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ReadRawBytes`:

Beispiel 1:

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num float := 0.2;
VAR string answer;

ClearRawBytes raw_data_out;
PackDNHeader "10", "20 1D 24 01 30 64", raw_data_out;
PackRawBytes float, raw_data_out, (RawBytesLen(raw_data_out)+1)
     \Float4;

Open "/FC1:/dsqc328_1", io_device \Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in \Time:=1;
Close io_device;
UnpackRawBytes raw_data_in, 1, answer \ASCII:=10;
```

In diesem Beispiel wird `raw_data_out` gelöscht und dann mit einem DeviceNet-Header und dem Float-Wert 0.2 gepackt.

Ein Gerät `"/FC1:/dsqc328_1"` wird geöffnet und die aktuellen gültigen Daten in `raw_data_out` werden auf das Gerät geschrieben. Dann wartet das Programm höchstens `1` Sekunde auf das Lesen von dem Gerät, das in `raw_data_in` gespeichert ist.

Nachdem das Gerät `"/FC1:/dsqc328_1"` geschlossen wurde, werden die gelesenen Daten als Zeichenfolge entpackt und in `answer` gespeichert.

Argumente

```
ReadRawBytes IODevice RawData [\Time]
```

`IODevice`

Datentyp: `iodev`

`IODevice` ist der Bezeichner für das Gerät, von dem die Daten gelesen werden sollen.

`RawData`

Datentyp: `rawbytes`

`RawData` ist der Datencontainer, in dem die vom Gerät `IODevice` gelesenen Daten ab Index 1 gespeichert werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.193 ReadRawBytes - Lesen der rawbytes-Daten

RobotWare - OS

Fortsetzung

`[\Time]`

Datentyp: `num`

Die maximale Dauer für den Lesevorgang (Timeout) in Sekunden (Auflösung 0,001 s). Wenn dieses Argument nicht angegeben wird, beträgt die maximale Dauer 60 Sekunden. Verwenden Sie für eine unendliche Wartezeit die vordefinierte Konstante `WAIT_MAX`.

Wenn diese Zeit abläuft, bevor der Lesevorgang abgeschlossen ist, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_DEV_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Die Timeout-Funktion ist auch während des Programmstopps in Gebrauch und wird vom RAPID-Programm beim Programmstart benachrichtigt.

Programmabarbeitung

Während der Programmabarbeitung werden Daten von dem mit `IODevice` angegebenen Gerät gelesen.

Wird `WriteRawBytes` für Feldbusbefehle wie DeviceNet verwendet, sendet der Feldbus immer eine Antwort. Die Antwort muss in RAPID mit der `ReadRawBytes`-Instruktion behandelt werden.

Die aktuelle Länge gültiger Bytes in der Variablen `RawData` wird auf die Anzahl gelesener Bytes gesetzt. Die Daten beginnen in `RawData` am Index 1.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Falscher Zugriff auf eine Datei
`ERR_DEV_MAXTIME`	Timeout bei Abarbeitung der Instruktion.
`ERR_RANYBIN_EOF`	Dateiende erkannt, bevor alle Bytes in der Instruktion `ReadRawBytes` gelesen wurden.

Syntax

```
ReadRawBytes
  [IODevice ':=' ] < variable (VAR) of iodev>' ,'
  [RawData ':=' ] < variable (VAR) of rawbytes> ','
  [ '\' Time ':=' < expression (IN) of num>] ';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.194 ReadVarArr - Mehrere Variablen von einem Sensorgerät ablesen

Verwendung

`ReadVarArr` wird verwendet, um gleichzeitig bis zu sechs Variablen von einem Sensorgerät abzulesen. Das Ergebnis stammt von der gleichen Abtastung.

Der Sensor muss über die RobotWare-Option *Sensor Interface* konfiguriert werden und kommunizieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ReadVarArr`.

Beispiel 1:

```
CONST num xcoord := 8;
CONST num ycoord := 9;
CONST num zcoord := 10;
VAR pos sensorpos;
VAR sensorvardata readdata{4};

! Connect to the sensor device “sen1:” (defined in sio.cfg).
SenDevice "sen1:";

! Read a cartesian position from the sensor.
readdata{1} := [xcoord, 2, false, 1, 0];
readdata{2} := [ycoord, 2, false, 1, 0];
readdata{3} := [zcoord, 2, false, 1, 0];
! A varNumber of -1 will be ignored
readdata{4} := [-1, 2, false, 1, 0];

ReadVarArr “sen1”, readdata;
sensorpos.x := DnumToNum(readdata{1}.value);
sensorpos.y := DnumToNum(readdata{2}.value);
sensorpos.z := DnumToNum(readdata{3}.value);
```

Das Beispiel zeigt das gleichzeitige Ablesen von drei Variablen. Das Ablesen erfolgt zur selben Zeit und stammt von derselben Abtastung von dem Sensor.

Argumente

```
ReadVarArr Device, Data, [\TaskName]
```

`Device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`Data`

Datentyp: `sensorvardata`

Eine Datenfeldvariable, die eine Datendefinition der Variablen bezeichnet, die gelesen werden. Der Ergebniswert der Ablesung wird in dieser Definition ebenfalls zurückgegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[ \TaskName ]
```

Datentyp: `string`

Das Argument `TaskName` ermöglicht, auf Geräte in anderen RAPID-Tasks zuzugreifen.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
SEN_NO_MEAS	Messfehler
SEN_NOREADY	Sensor kann Befehl nicht handhaben
SEN_GENERRO	Allgemeiner Sensorfehler
SEN_BUSY	Sensor belegt
SEN_UNKNOWN	Unbekannter Sensor
SEN_EXALARM	Externer Sensorfehler
SEN_CAALARM	Interner Sensorfehler
SEN_TEMP	Sensortemperaturfehler
SEN_VALUE	Unzulässiger Kommunikationswert
SEN_CAMCHECK	Sensorprüffehler
SEN_TIMEOUT	Kommunikationsfehler

Syntax

```
ReadVarArr
  [Device ':='] <expression(IN) of string>','
  [Data ':='] <array variable {*} (INOUT) of sensorvardata>','
  ['\' TaskName ':=' <expression (IN) of string>]';'
```

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Schreiben mehrerer Variablen auf ein Gerät	*WriteVarArr - Schreiben mehrerer Variablen auf ein Sensorgerät auf Seite 1135*
Schreiben einer Sensorvariablen	*WriteVar – Variable schreiben auf Seite 1132*
Lesen einer Sensorvariablen	*ReadVar - Lesen einer Variablen von einem Gerät auf Seite 1474*
Schreiben eines Sensordatenblocks	*WriteBlock – Datenblock auf Gerät schreiben auf Seite 1121*
Lesen eines Sensordatenblocks	*ReadBlock - Lesen eines Datenblocks von einem Gerät auf Seite 584*
Mehrere variable Konfigurationsdaten für die Sensorschnittstelle	*sensorvardata - Mehrere variable Konfigurationsdaten für die Sensorschnittstelle auf Seite 1785*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.194 ReadVarArr - Mehrere Variablen von einem Sensorgerät ablesen

Sensor Interface

Fortsetzung

Information über	Siehe
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.195 RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen

Verwendung

`RemoveAllCyclicBool` wird verwendet, um die zyklische Auswertung aller Cyclic bool-Bedingungen der Task zu entfernen, die die Instruktion ausführt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RemoveAllCyclicBool`.

Beispiel 1:

```
PERS bool cyclicflag1;
TASK PERS bool cyclicflag2;
PERS bool mypersbool:=FALSE;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  SetupCyclicBool cyclicflag2, di3 AND di4 AND mypersbool=FALSE;
  ...
  RemoveAllCyclicBool;
  ...
```

Zuerst werden zwei zyklische Auswertungen konfiguriert, dann werden später beide entfernt.

Beispiel 2:

```
PERS bool cyclicflag1;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  ...
  RemoveAllCyclicBool;
UNDO
  RemoveAllCyclicBool;
ENDPROC
```

Alle zyklischen Auswertungen werden entfernt, wenn der Programmzeiger auf Main gesetzt wird.

Das gleiche Verhalten kann in den Systemparametern konfiguriert werden, ohne eine `UNDO`-Behandlung durchzuführen, siehe *Technisches Referenzhandbuch - Systemparameter*.

Argumente

```
RemoveAllCyclicBool [\AllTasks]
```

`[\AllTasks]`

Datentyp: `switch`

Dieses Argument wird verwendet, um die zyklische Auswertung für alle Tasks zu entfernen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.195 RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen

RobotWare - OS

Fortsetzung

Programmabarbeitung

Das Verhalten der Cyclic bool-Funktion ist konfigurierbar. Weitere Informationen finden Sie unter *Anwendungshandbuch - Steuerungssoftware IRC5* und *Technisches Referenzhandbuch - Systemparameter*.

Syntax

```
RemoveAllCyclicBool
  ['\'AllTasks] ';'
```

Weitere Informationen

Information über	Siehe
Konfiguration einer Cyclic bool-Bedingung	*SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung auf Seite 722*
Entfernen einer Cyclic bool-Bedingung	*RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung auf Seite 601*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.196 RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung

Verwendung

`RemoveCyclicBool` wird verwendet, um die zyklische Auswertung einer Cyclic bool-Bedingung zu entfernen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `RemoveCyclicBool`.

Beispiel 1:

```
PERS bool cyclicflag1;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  ...
  RemoveCyclicBool cyclicflag1;
  ...
```

Zunächst wird eine zyklische Auswertung eingerichtet, die dann später entfernt wird.

Beispiel 2:

```
PERS bool cyclicflag1;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  ...
  RemoveCyclicBool "cyclicflag1";
  ...
```

Zunächst wird eine zyklische Auswertung konfiguriert, später wird sie dann entfernt, indem der Name der persistenten booleschen Variable verwendet wird.

Argumente

```
RemoveCyclicBool Flag | Name
```

`Flag`

Datentyp: `bool`

Die persistente boolesche Variable, die den Wert der logischen Bedingung speichert.

`Name`

Datentyp: `string`

Der Name der persistenten booleschen Variable, die den Wert des logischen Zustands speichert.

Programmabarbeitung

Das Verhalten der Cyclic bool-Funktion ist konfigurierbar. Weitere Informationen finden Sie unter *Anwendungshandbuch - Steuerungssoftware IRC5* und *Technisches Referenzhandbuch - Systemparameter*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.196 RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung

RobotWare - OS

Fortsetzung

Syntax

```
RemoveCyclicBool
  [ Flag ':=' ] <persistent (PERS) of bool>
  | [ Name ':=' ] <expression (IN) of string> ';'
```

Weitere Informationen

Information über	Siehe
Prüfen, ob eine persistente Variable eine Cyclic bool ist	*IsCyclicBool - Prüft, ob eine persistente Variable eine Cyclic bool ist auf Seite 1369*
Konfiguration einer Cyclic bool-Bedingung	*SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung auf Seite 722*
Alle Cyclic bool-Bedingungen entfernen	*RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen auf Seite 599*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.197 RemoveDir - Löschen eines Verzeichnisses

Verwendung

`RemoveDir` wird verwendet, um ein Verzeichnis zu löschen.

Der Benutzer muss für das Verzeichnis über eine Schreib- und Ausführungserlaubnis verfügen und das Verzeichnis muss leer sein.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RemoveDir`:

Beispiel 1:

```
RemoveDir "HOME:/mydir";
```

In diesem Beispiel wird das Verzeichnis `mydir` unter `HOME:` gelöscht.

Argumente

```
RemoveDir Path
```

`Path`

Datentyp: `string`

Der Name des zu löschenden Verzeichnisses mit dem vollständigen oder relativen Pfad.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Das Verzeichnis existiert nicht, ist nicht leer, oder der Benutzer verfügt über keine Schreib- und Ausführungsberechtigung.

Syntax

```
RemoveDir
[ Path':=' ] < expression (IN) of string>';'
```

Weitere Informationen

Information über	Siehe
Verzeichnis	*dir – Dateiverzeichnisstruktur auf Seite 1689*
Öffnen eines Verzeichnisses	*OpenDir - Öffnen eines Verzeichnisses auf Seite 497*
Lesen eines Verzeichnisses	*ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis auf Seite 1459*
Schließen eines Verzeichnisses	*CloseDir - Schließen eines Verzeichnisses auf Seite 153*
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.197 RemoveDir - Löschen eines Verzeichnisses

RobotWare - OS

Fortsetzung

Information über	Siehe
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.198 RemoveFile - Löschen einer Datei

Verwendung

`RemoveFile` wird verwendet, um Dateien zu löschen. Der Benutzer muss für das Verzeichnis, in dem sich die Datei befindet, über Schreib- und Ausführungserlaubnis und für die Datei selbst über Schreiberlaubnis verfügen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RemoveFile`:

Beispiel 1:

```
RemoveFile "HOME:/mydir/myfile.log";
```

In diesem Beispiel wird die Datei `myfile.log` im Verzeichnis `mydir` auf dem Speichermedium `HOME:` gelöscht.

Argumente

```
RemoveFile Path
```

`Path`

Datentyp: string

Der Name der zu löschenden Datei im Dateisystem mit dem vollständigen oder relativen Pfad.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Falscher Zugriff auf eine Datei.

Syntax

```
RemoveFile
[ Path':=' ] < expression (IN) of string>';'
```

Weitere Informationen

Information über	Siehe
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.198 RemoveFile - Löschen einer Datei

RobotWare - OS

Fortsetzung

Information über	Siehe
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.199 RemoveSuperv Bedingung für ein Signal entfernen

Verwendung

`RemoveSuperv` wird verwendet, um Bedingungen zu entfernen, die von der Überwachung durch `SetupSuperv` hinzugefügt wurden.

Grundlegendes Beispiel

```
PROC main()
  InitSuperv;
  SetupSuperv diWR_EST, ACT, SUPERV_MAIN \ErrIndSig:= do_WR_Sup;
  SetupSuperv diGA_EST, ACT, SUPERV_MAIN;
  CapL p2, v100, cdata1, weavestart, weave,fine, tWeldGun;
  RemoveSuperv di_Arc_Sup, ACT, SUPERV_START;
ENDPROC
```

Entfernt das Signal *di_Arc_Sup* aus der Liste START:

Argumente

```
RemoveSuperv Signal Condition Listtype
```

Signal

Datentyp: `signaldi`

Signal, das aus der Überwachungsliste entfernt werden soll.

Condition

Datentyp: `num`

Der Name, der eine der folgenden verfügbaren Bedingungen repräsentiert:

ACT:	Wird für die Statusüberwachung verwendet. Erwarteter Signalstatus während der Überwachung: aktiv. Wenn das Signal passiv wird, löst es die Überwachung aus.
PAS:	Wird für die Statusüberwachung verwendet. Erwarteter Signalstatus während der Überwachung: passiv. Wenn das Signal aktiv wird, löst es die Überwachung aus.
POS_EDGE:	Wird für Handshake-Überwachung verwendet. Erwarteter Signalstatus am Ende der Überwachung: aktiv. Wenn das Signal innerhalb des gewählten Zeitüberschreitung nicht aktiv wird, löst es die Überwachung aus.
NEG_EDGE:	Wird für Handshake-Überwachung verwendet. Erwarteter Signalstatus am Ende der Überwachung: passiv. Wenn das Signal innerhalb des gewählten Zeitüberschreitung nicht passiv wird, löst es die Überwachung aus.

Listtype

Datentyp: `num`

Der Name, der die Anzahl der verschiedenen Listen (beispielsweise Phasen des Prozesses) repräsentiert:

- SUPERV_PRE
- SUPERV_PRE_START
- SUPERV_END_PRE
- SUPERV_START

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.199 RemoveSuperv Bedingung für ein Signal entfernen

Continuous Application Platform (CAP)
Fortsetzung

- SUPERV_MAIN
- SUPERV_END_MAIN
- SUPERV_START_POST1
- SUPERV_POST1
- SUPERV_END_POST1
- SUPERV_START_POST2
- SUPERV_POST2
- SUPERV_END_POST2

Syntax

```
RemoveSuperv
  [Signal ':='] < variable (VAR) of signaldi > ','
  [Condition ':='] < variable (IN) of num > ','
  [Listtype ':='] < variable (IN) of num >';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instruktion `InitSuperv`	*InitSuperv - Zurücksetzen der gesamten CAP-Überwachung auf Seite 284*
Instruktion `SetupSuperv`	*SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP auf Seite 725*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.200 RenameFile - Umbenennen einer Datei

Verwendung

`RenameFile` wird verwendet, um einer bereits vorhandenen Datei einen neuen Namen zu geben. Die Instruktion kann darüber hinaus verwendet werden, um eine Datei aus einem in ein anderes Verzeichnis zu verschieben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RenameFile`:

Beispiel 1:

```
RenameFile "HOME:/myfile", "HOME:/yourfile";
```

Die Datei `myfile` wird in `yourfile` umbenannt.

```
RenameFile "HOME:/myfile", "HOME:/mydir/yourfile";
```

Die Datei `myfile` wird in `yourfile` umbenannt und in das Verzeichnis `mydir` verschoben.

Argumente

```
RenameFile OldPath NewPath
```

`OldPath`

Datentyp:`string`

Der vollständige Pfad der umzubenennenden Datei.

`NewPath`

Datentyp:`string`

Der komplette Verzeichnispfad der umbenannten Datei.

Programmabarbeitung

Die unter `OldPath` angegebene Datei erhält den in `NewPath` angegebenen Namen. Falls sich der unter `NewPath` angegebene Pfad von dem unter `OldPath` angegebenen Pfad unterscheidet, wird die Datei darüber hinaus in das neue Verzeichnis verschoben.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Die in `OldPath` angegebene Datei existiert nicht.
`ERR_FILEEXIST`	Die unter `NewPath` angegebene Datei ist bereits vorhanden.

Syntax

```
RenameFile
[ OldPath' :=' ] < expression (IN) of string > ','
[ NewPath' :=' ] < expression (IN) of string >';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.200 RenameFile - Umbenennen einer Datei

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.201 Reset - Setzt ein digitales Ausgangssignal zurück

Verwendung

`Reset` wird verwendet, um den Wert eines digitalen Ausgangssignals auf null zu setzen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Reset`:

Beispiel 1:

```
Reset do15;
```

Das Signal `do15` wird auf 0 gesetzt.

Beispiel 2:

```
Reset weld;
```

Das Signal `weld` wird auf 0 gesetzt.

Argumente

```
Reset Signal
```

`Signal`

Datentyp: `signaldo`

Der Name des Signals, das auf 0 gesetzt werden soll.

Programmabarbeitung

Der wahre Wert hängt von der Konfiguration des Signals ab. Wenn das Signal in den Systemparametern invertiert wird, wird der physische Kanal durch diese Instruktion auf 1 gesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
Reset
  [Signal ':='] <variable (VAR) of signaldo>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.201 Reset - Setzt ein digitales Ausgangssignal zurück

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Setzen eines digitalen Ausgangssignals	*Set - Setzen eines digitalen Ausgangssignals auf Seite 695*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.202 ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse

Verwendung

`ResetAxisDistance` wird verwendet, um die Informationen über die zurückgelegte Strecke für die Achse zurückzusetzen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ResetAxisDistance`.

Beispiel 1:

```
ResetAxisDistance Track,1;
```

Die Informationen über die zurückgelegte Strecke für die Achse 1 an der mechanischen Einheit `Track` werden zurückgesetzt.

Beispiel 2:

```
PERS dnum distanceLimit := 1000;

PROC main()
  IF GetAxisDistance(Track,1) > distanceLimit THEN
    ErrWrite \I, ”Distance counter limit reached”, ”Distance counter
         limit for Track has been reached.”;
    DoMaintenance();
  ENDIF
ENDPROC

PROC DoMaintenance()
  ...
  ResetAxisDistance Track, 1;
  ErrWrite \I, ”Distance counter reset”, ”Distance counter for
       Track has been reset.”;
ENDPROC
```

Das Beispiel beschreibt, wie `ResetAxisDistance` zusammen mit `GetAxisDistance` verwendet wird, um zu überprüfen, ob es Zeit für die Wartung einer Achse ist.

Argumente

```
ResetAxisDistance MechUnit AxisNo
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`AxisNo`

Datentyp: `num`

Die Nummer der Achse, für welche die zurückgelegte Strecke zurückgesetzt werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.202 ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse

RobotWare - OS

Fortsetzung

Programmabarbeitung

Setzt die Informationen über die zurückgelegte Strecke für die ausgewählte Achse zurück.

Syntax

```
ResetAxisDistance
  [MechUnit ':='] < variable (VAR) of mecunit > ','
  [AxisNo ':='] < variable (VAR) of num > ';'
```

Weitere Informationen

Information über	Siehe
ResetAxisMoveTime	*ResetAxisMoveTime - Zurücksetzen des Bewegungszeitzählers der Achse auf Seite 615*
GetAxisDistance	*GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse auf Seite 1313*
GetAxisMoveTime	*GetAxisMoveTime - Abrufen des Bewegungszeitzählers der Achse auf Seite 1315*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.203 ResetAxisMoveTime - Zurücksetzen des Bewegungszeitzählers der Achse

Verwendung

`ResetAxisMoveTime` wird verwendet, um die Bewegungszeitinformationen für die Achse zurückzusetzen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `ResetAxisMoveTime`.

Beispiel 1:

```
ResetAxisMoveTime Track,1;
```

Die Bewegungszeitinformationen für die Achse 1 an der mechanischen Einheit `Track` werden zurückgesetzt.

Beispiel 2:

```
PERS dnum timeLimit := 1000;

PROC main()
  IF GetAxisMoveTime(Track,1) > timeLimit THEN
    ErrWrite \I, ”Time counter limit reached”, ”Time counter limit
         for Track has been reached.”;
    DoMaintenance();
  ENDIF
ENDPROC

PROC DoMaintenance()
  ...
  ResetAxisMoveTime Track, 1;
  ErrWrite \I, ”Time counter reset”, ”Time counter for Track has
       been reset.”;
ENDPROC
```

Das Beispiel beschreibt, wie `ResetAxisMoveTime` zusammen mit `GetAxisMoveTime` verwendet wird, um zu überprüfen, ob es Zeit für die Wartung einer Achse ist.

Argumente

```
ResetAxisMoveTime MechUnit AxisNo
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`AxisNo`

Datentyp: `num`

Die Nummer der Achse, für welche die Bewegungszeit zurückgesetzt werden soll.

Programmabarbeitung

Setzt die Bewegungszeitinformationen für die ausgewählte Achse zurück.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.203 ResetAxisMoveTime - Zurücksetzen des Bewegungszeitzählers der Achse

RobotWare - OS

Fortsetzung

Syntax

```
ResetAxisMoveTime
  [MechUnit ':='] < variable (VAR) of mecunit > ','
  [AxisNo ':='] < variable (VAR) of num > ';'
```

Weitere Informationen

Information über	Siehe
ResetAxisDistance	*ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse auf Seite 613*
GetAxisDistance	*GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse auf Seite 1313*
GetAxisMoveTime	*GetAxisMoveTime - Abrufen des Bewegungszeitzählers der Achse auf Seite 1315*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.204 ResetPPMoved - Zurücksetzen des Status für den im Einrichtbetrieb verschobenen Programmzeiger

Verwendung

`ResetPPMoved` Setzt den Status für den im Einrichtbetrieb verschobenen Programmzeiger zurück. `PPMovedInManMode` gibt TRUE zurück, wenn der Benutzer den Programmzeiger verschoben hat, während sich die Steuerung im Einrichtbetrieb befand, d. h. der Betriebsartenwahlschalter war in der Stellung Einrichtbetrieb bei reduzierter Geschwindigkeit oder Einrichtbetrieb bei voller Geschwindigkeit. Der Status des verschobenen Programmzeigers wird mit der Instruktion `ResetPPMoved` oder durch Schalten des Betriebsartenwahlschalters von Automatikbetrieb auf Einrichtbetrieb zurückgesetzt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ResetPPMoved`:

Beispiel 1:

```
IF PPMovedInManMode() THEN
  WarnUserOfPPMovement;
  ! DO THIS ONLY ONCE
  ResetPPMoved;
  DoJob;
ELSE
  DoJob;
ENDIF
```

Programmabarbeitung

Setzt den Status für den im Einrichtbetrieb verschobenen Programmzeiger zurück.

Syntax

```
ResetPPMoved';'
```

Weitere Informationen

Information über	Siehe
Überprüfen, ob der Programmzeiger im Einrichtbetrieb verschoben wurde	*PPMovedInManMode - Überprüfen, ob der Programmzeiger im Einrichtbetrieb verschoben wurde auf Seite 1447*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.205 ResetRetryCount - Zurücksetzen der Anzahl an Wiederholungen

Verwendung

`ResetRetryCount` wird verwendet, um die Anzahl der Wiederholungen durch eine Fehlerbehandlungsroutine zurückzusetzen. Die maximale Anzahl der möglichen Wiederholungen ist in der Konfiguration definiert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ResetRetryCount`:

Beispiel 1:

```
VAR num myretries := 0;
...
ERROR
  IF myretries > 2 THEN
    ResetRetryCount;
    myretries := 0;
    TRYNEXT;
  ENDIF
  myretries:= myretries + 1;
  RETRY;
...
```

Mit diesem Programm wird die fehlerhafte Instruktion drei Mal wiederholt, und dann wird die nächste Instruktion ausgeführt. Der interne Wiederholungszähler des Systems wird zurückgesetzt, bevor die nächste Instruktion ausgeführt wird (auch wenn dies durch das System bei `TRYNEXT` erfolgt).

Programmabarbeitung

Für jedes RETRY durch eine Fehlerbehandlungsroutine überprüft ein interner Systemzähler, ob die maximale Anzahl an Wiederholungen, die in der Konfiguration angegeben ist, überschritten wird. Durch die Ausführung der Instruktion `ResetRetryCount` wird der Zähler zurückgesetzt und die maximale Anzahl an Wiederholungen kann erneut ausgeführt werden.

Syntax

```
ResetRetryCount ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlungen	*Technisches Referenzhandbuch - RAPID Overview*
Wiederaufnahme der Abarbeitung nach einem Fehler	*RETRY - Wiederaufnahme der Abarbeitung nach einem Fehler auf Seite 622*
Konfigurieren der maximalen Anzahl an Wiederholversuchen	*Technisches Referenzhandbuch - Systemparameter*
Restliche Anzahl an Wiederholungen	*RemainingRetries Restliche Anzahl an Wiederholungen auf Seite 1478*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.206 ResetTorqueMargin - Die geringste Drehmomentspanne zurückstellen

Verwendung

`ResetTorqueMargin` wird verwendet, um die geringste Drehmomentspanne zurückzusetzen, um eine neue Messung starten zu können.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `ResetTorqueMargin`.

Beispiel 1:

```
ResetTorqueMargin \AxisNo:=3;
! starts a new measurement for axis 3;
```

Argumente

```
ResetTorqueMargin [\MecUnit ] [ \AxisNo ]
```

`[\MecUnit ]`

Datentyp: `mecunit`

Der Name der mechanischen Einheit, für die eine Achse Reset werden soll. Wenn dieses Argument weggelassen wird, wird die Achse für den angeschlossenen Roboter Reset.

`[\AxisNo]`

Datentyp: `num`

Die Nummer der zurückzusetzenden Achse (1 bis 6).

Ist keine Achse angegeben, werden alle Achsen für die mechanische Einheit zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_AXIS_PAR`	Parameter "Achse" in der Funktion ist falsch.

Syntax

```
ResetTorqueMargin '('
  [ '\' MechUnit ':=' ] < variable (VAR) of mecunit > ','
  [ '\' AxisNo ':=' < expression (IN) of num > ] ')'
```

Weitere Informationen

Information über	Siehe
Erhalten der Drehmomentspanne	*GetTorqueMargin - Liest die geringste Drehmomentspanne ab auf Seite 1342*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.207 RestoPath - Wiederherstellen der Bahn nach einem Interrupt

Verwendung

`RestoPath` wird verwendet, um eine Bahn wiederherzustellen, die in einem früheren Stadium über die Instruktion `StorePath` gespeichert wurde.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RestoPath`:

Siehe auch *Weitere Beispiele* weiter unten.

Beispiel 1:

```
RestoPath;
```

Stellt die Bahn wieder her, die zu einem früheren Zeitpunkt mithilfe von `StorePath` gespeichert wurde.

Programmabarbeitung

Die aktuelle Bewegungsbahn des Roboters und der externen Achsen wird gelöscht und die zuvor über `StorePath` gespeicherte Bahn wird wiederhergestellt. Es erfolgt jedoch erst dann wieder eine Bewegung, wenn die Instruktion `StartMove` abgearbeitet wird oder mit `RETRY` die Rückkehr aus einer Fehlerbehandlungsroutine erfolgt.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `RestoPath` sind unten aufgeführt.

Beispiel 1:

```
ArcL p100, v100, seam1, weld5 \Weave:=weave1, z10, gun1;
...
ERROR
  IF ERRNO=AW_WELD_ERR THEN
    gun_cleaning;
    StartMoveRetry;
  ENDIF
...
PROC gun_cleaning()
  VAR robtarget p1;
  StorePath;
  p1 := CRobT();
  MoveL pclean, v100, fine, gun1;
  ...
  MoveL p1, v100, fine, gun1;
  RestoPath;
ENDPROC
```

Bei Auftreten eines Schweißfehlers fährt die Programmabarbeitung in der Fehlerbehandlung der Routine fort, die wiederum `gun_cleaning` aufruft. Die Bewegungsbahn, die gerade abgearbeitet wird, wird dann gespeichert und der

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Roboter bewegt sich an die Position `pclean`, wo der Fehler behoben wird. Anschließend kehrt der Roboter an die Position `p1` zurück, an der der Fehler aufgetreten ist, und speichert erneut die Originalbewegung. Der Schweißvorgang startet dann automatisch neu, d. h., der Roboter bewegt sich zunächst auf der Bahn etwas zurück, bevor das Schweißen beginnt und die normale Programmabarbeitung fortfährt.

Einschränkungen

Nur die Bewegungsbahndaten werden mit der Instruktion `StorePath` gespeichert. Wenn der Anwender Bewegungen auf der neuen Bahnebene anfordern will, muss die aktuelle Stoppposition direkt nach `StorePath` gespeichert werden. Vor `RestoPath` muss dann zuerst eine Bewegung an die gespeicherte Stoppposition auf der Bahn erfolgen.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`RestoPath` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

Syntax

```
RestoPath';'
```

Weitere Informationen

Information über	Siehe
Speichern von Bahnen	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Weitere Beispiele	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *SyncMoveSuspend - Festlegen von unabhängigen teilkoordinierten Bewegungen auf Seite 871*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.208 RETRY - Wiederaufnahme der Abarbeitung nach einem Fehler

Verwendung

Die Instruktion RETRY wird verwendet, um die Programmabarbeitung nach einem Fehler wieder aufzunehmen, beginnend mit der Instruktion, die den Fehler verursacht hat.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion RETRY:

Beispiel 1:

```
reg2 := reg3/reg4;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    reg4 :=1;
    RETRY;
  ENDIF
```

Es wird versucht, reg3 durch reg 4 zu dividieren. Wenn reg4 gleich 0 ist (Division durch null), erfolgt ein Sprung zur Fehlerbehandlungsroutine, die reg4 initialisiert. Die Instruktion RETRY wird dann zum Sprung aus der Fehlerbehandlungsroutine verwendet und es erfolgt ein weiterer Versuch, die Division auszuführen.

Programmabarbeitung

Die Programmabarbeitung fährt mit der Instruktion fort, die den Fehler verursacht hat.

Fehlerbehandlung

Wenn die maximale Anzahl an Wiederholversuchen (4) überschritten wird, stoppt die Programmabarbeitung mit einer Fehlermeldung. Die maximale Anzahl an Wiederholungen kann unter in Systemparametern (Typ *General RAPID*) konfiguriert werden.

Einschränkungen

Die Instruktion ist nur in der Fehlerbehandlung innerhalb einer Routine möglich. Wenn der Fehler mit einer RAISE-Instruktion erzeugt wurde, kann die Programmabarbeitung nicht mit der Instruktion RETRY neu gestartet werden. In diesem Fall muss die Instruktion TRYNEXT verwendet werden.

Syntax

```
RETRY ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlungen	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Konfigurieren der maximalen Anzahl an Wiederholversuchen	*Technisches Referenzhandbuch - Systemparameter*
Fortfahren mit der nächsten Instruktion	*TRYNEXT - Überspringen einer fehlerverursachenden Instruktion auf Seite 1002*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.209 RETURN - Beenden der Abarbeitung einer Routine

Verwendung

RETURN wird verwendet, um die Abarbeitung einer Routine zu beenden. Wenn es sich bei der Routine um eine Funktion handelt, wird auch der Wert der Funktion zurückgegeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion RETURN:

Beispiel 1:

```
errormessage;
Set do1;
...
PROC errormessage()
  IF di1=1 THEN
    RETURN;
  ENDIF
  TPWrite "Error";
ENDPROC
```

Die Prozedur errormessage wird aufgerufen. Wenn die Prozedur die Instruktion RETURN erreicht, kehrt die Programmabarbeitung zu der Instruktion zurück, die auf den Prozeduraufruf Set do 1 folgt.

Beispiel 2:

```
FUNC num abs_value(num value)
  IF value<0 THEN
    RETURN -value;
  ELSE
    RETURN value;
  ENDIF
ENDFUNC
```

Die Funktion gibt den absoluten Wert einer Zahl zurück.

Argumente

```
RETURN [ Return value ]
```

Return value

Datentyp: Entsprechend der Funktionsdeklaration.

Der Rückgabewert einer Funktion.

Der Rückgabewert muss in einer RETURN-Instruktion angegeben werden, die sich in einer Funktion befindet.

Wenn sich die Instruktion in einer Prozedur oder Interrupt-Routine befindet, kann kein Rückgabewert angegeben werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Das Resultat der Instruktion RETURN kann variieren, je nach der Art der Routine, in der sie verwendet wird:

- Hauptroutine: Bei einem Programm mit dem Abarbeitungsmodus „Einzelzyklus“ wird das Programm angehalten. Andernfalls fährt die Programmabarbeitung mit der ersten Instruktion in der Hauptroutine fort.
- Prozedur: Die Programmabarbeitung fährt mit der Instruktion fort, die auf den Prozeduraufruf folgt.
- Funktion: Gibt den Wert der Funktion zurück.
- Trap-Routine: Die Programmabarbeitung wird an der Stelle fortgesetzt, an der der Interrupt aufgetreten ist.
- Fehlerbehandlungsroutine in einer Prozedur: Die Programmabarbeitung fährt mit der Routine fort, von der aus die Routine mit der Fehlerbehandlung aufgerufen wurde (mit der Instruktion, die dem Prozeduraufruf folgt).
- Fehlerbehandlungsroutine in einer Funktion: Der Wert der Funktion wird zurückgegeben.

Syntax

```
RETURN [ <expression> ]';'
```

Weitere Informationen

Information über	Siehe
Funktionen und Prozeduren	*Technisches Referenzhandbuch - RAPID Overview*
Interrupt-Routinen	*Technisches Referenzhandbuch - RAPID Overview*
Fehlerbehandlungen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.210 Rewind - Zurücksetzen des Dateizeigers

Verwendung

`Rewind` setzt den Dateizeiger an den Beginn der Datei.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Rewind`:

Beispiel 1:

```
Rewind iodev1;
```

In Datei `iodev1` wird der Dateizeiger an den Beginn der Datei gesetzt.

Argumente

```
Rewind IODevice
```

`IODevice`

Datentyp: `iodev`

Name (Verweis) der Datei, deren Dateizeiger zurück an den Anfang gesetzt wird.

Programmabarbeitung

Der Dateizeiger in der angegebenen Datei wird auf den Anfang gesetzt.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Falscher Zugriff auf eine Datei.

Einschränkungen

Wenn die verwendete Datei mit einem `\Bin`- oder `\Bin \Append`-Schalter geöffnet wurde , ist `Rewind` vor jedem Typ einer `Write`-Instruktion ohne Auswirkung. Der Schreibvorgang erfolgt am Ende der Datei.

Syntax

```
Rewind [IODevice ':='] <variable (VAR) of iodev>';'
```

Weitere Informationen

Information über	Siehe
Öffnen usw. von Dateien	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.211 RMQEmptyQueue - Leere RAPID Message Queue

Verwendung

`RMQEmptyQueue` leert die RAPID Message Queue (RMQ) in der Task, welche die Instruktion abarbeitet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RMQEmptyQueue`:

Beispiel

```
RMQEmptyQueue;
```

Die Instruktion `RMQEmptyQueue` entfernt in der ausführenden Task alle Meldungen aus der RMQ.

Programmabarbeitung

Die der ausführenden Task eigene RAPID Message Queue wird geleert. Die Instruktion kann auf allen Abarbeitungsebenen verwendet werden.

Einschränkungen

`RMQEmptyQueue` leert nur die RAPID Message Queue in der Task, welche die Instruktion ausführt. Alle anderen RAPID Message Queues bleiben unverändert.

Syntax

```
RMQEmptyQueue ';'
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
`rmqmessage` Datentyp	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*.
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*.
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*.
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*.
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*.
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*.
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.211 RMQEmptyQueue - Leere RAPID Message Queue

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

Information über	Siehe
Empfang einer Meldung von der RMQ	*RMQReadWait – Rückgabe der Meldung von der RMQ auf Seite 640*.
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.212 RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens

Verwendung

`RMQFindSlot` (*RAPID Message Queue Find Slot*) wird verwendet, um die Slot-Identität einer für eine RAPID-Task konfigurierten RMQ oder eines Robot Application Builder-Clients zu suchen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RMQFindSlot`:

Beispiel 1:

```
VAR rmqslot myrmqslot;
RMQFindSlot myrmqslot, "RMQ_T_ROB2";
```

Ruft die Identitätsnummer der RMQ`"RMQ_T_ROB2"` ab, die für die RAPID-Task `"T_ROB2"` konfiguriert ist.

Argumente

```
RMQFindSlot Slot Name
```

`Slot`

Datentyp: `rmqslot`

Die Variable, in der der numerische Bezeichner zurückgegeben wird.

`Name`

Datentyp: `string`

Der Name des Clients, dessen Identitätsnummer gesucht werden soll. Dabei muss die Groß- und Kleinschreibung beachtet werden. Wenn die RAPID-Task T_ROB1 bezeichnet wird und der Name RMQ_t_rob1 für die RMQ verwendet wird, führt das zu einem Fehler (siehe Abschnitt Fehlerbehandlung im Anschluss).

Programmabarbeitung

Mit der `RMQFindSlot`-Instruktion wird die Slot-Identität für eine benannte RMQ oder einen benannten Robot Application Builder-Client gesucht.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_RMQ_NAME`	Der angegebene Slot-Name ist ungültig oder wurde nicht gefunden.

Syntax

```
RMQFindSlot
  [ Slot ':=' ] < variable (VAR) of rmqslot > ','
  [ Name ':=' ] < expression (IN) of string > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.212 RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*
`RMQ Slot`	*rmqslot - Identitätsnummer eines RMQ-Clients auf Seite 1776*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.213 RMQGetMessage - Abrufen einer RMQ-Meldung

Verwendung

`RMQGetMessage` (*RAPID Message Queue Get Message*) wird verwendet, um die erste RMQ-Meldung aus der Warteschlange für die aktuelle Programmtask abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RMQGetMessage`:

Siehe auch *Weitere Beispiele auf Seite 632*.

Beispiel 1:

```
TRAP msghandler
  VAR rmqmessage myrmqmsg;
  RMQGetMessage myrmqmsg;
  ...
ENDTRAP
```

In der TRAP -Routine `msghandler` wird die `rmqmessage` aus der RMQ abgerufen und in die Variable `myrmqmsg` kopiert.

Argumente

```
RMQGetMessage Message
```

`Message`

Datentyp: `rmqmessage`

Variable zum Speichern der RMQ-Meldung.

Die maximale Datenmenge, die in einer `rmqmessage` empfangen werden kann, beträgt ca. 3000 Byte.

Programmabarbeitung

Mit der Instruktion `RMQGetMessage` wird die erste Meldung aus der Warteschlange der Task abgerufen, die die Instruktion ausführt.Wenn eine Meldung vorhanden ist, wird sie in die `Message`-Variable kopiert und dann aus der Warteschlange entfernt, um für neue Meldungen Platz zu schaffen. Die Instruktion wird nur auf der TRAP-Ebene unterstützt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_RMQ_NOMSG`	Gegenwärtig befindet sich keine Meldung in der Warteschlange. Dieser Fall kann eintreten, wenn `RMQGetMessage` zweimal in einer TRAP-Routine ausgeführt wird. Der Fehler kann auch erzeugt werden, wenn zwischen dem Aufruf der TRAP-Routine und der Ausführung der Instruktion `RMQGetMessage` ein Stromausfall auftritt. Bei einem Stromausfall gehen die Meldungen in der RMQ verloren.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.213 RMQGetMessage - Abrufen einer RMQ-Meldung
FlexPendant Interface, PC Interface, or Multitasking
Fortsetzung

Name	Fehlerursache
ERR_RMQ_INVMSG	Ungültige Meldung, wahrscheinlich von anderem Client als einer RAPID-Task gesendet. Das könnte beispielsweise passieren, wenn eine PC-Anwendung eine fehlerhafte Meldung sendet.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion RMQGetMessage sind unten aufgeführt.

Beispiel 1:

```
RECORD mydatatype
  int x;
  int y;
ENDRECORD

VAR intnum msgreceive;
VAR mydatatype mydata;

PROC main()
  ! Setup interrupt
  CONNECT msgreceive WITH msghandler;
  ! Order cyclic interrupt to occur for data type mydatatype
  IRMQMessage mydata, msgreceive;
  WHILE TRUE DO
    ! Performing cycle
    ...
  ENDWHILE
ENDPROC

TRAP msghandler
  VAR rmqmessage message;
  VAR rmqheader header;

  ! Get the RMQ message
  RMQGetMessage message;
  ! Copy RMQ header information
  RMQGetMsgHeader message \Header:=header;

  IF header.datatype = "mydatatype" AND header.ndim = 0 THEN
    ! Copy the data from the message
    RMQGetMsgData message, mydata;
  ELSE
    TPWrite "Received a type not handled or with wrong dimension";
  ENDIF
ENDTRAP
```

Beim Empfang einer neuen Meldung wird die TRAP-Routine msghandler ausgeführt, und die neue Meldung wird in die Variable message kopiert (Instruktion RMQGetMessage). Dann werden die RMQ-Headerdaten kopiert (Instruktion RMQGetMsgHeader). Wenn die Meldung den erwarteten Datentyp und die richtige

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dimension aufweist, werden die Daten in die Variable `mydata` kopiert (Instruktion `RMQGetMsgData`).

Einschränkungen

`RMQGetMessage` wird bei der Abarbeitung auf der Benutzerebene (d. h. in Serviceroutinen) und auf der normalen Abarbeitungsebene nicht unterstützt.

Die maximale Datenmenge, die in einer `rmqmessage` empfangen werden kann, beträgt ca. 3000 Byte.

Es wird empfohlen, Variablen vom Datentyp `rmqmessage` so häufig wie möglich wiederzuverwenden, um RAPID-Speicher zu sparen.

Syntax

```
RMQGetMessage
  [ Message ':=' ] < variable (VAR) of rmqmessage > ';'
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*
`RMQ Message`	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.214 RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung

Verwendung

`RMQGetMsgData` (*RAPID Message Queue Get Message Data*) wird verwendet, um die aktuellen Daten in der RMQ-Meldung abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RMQGetMsgData`:

Siehe auch *RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*.

Beispiel 1:

```
VAR rmqmessage myrmqmsg;
VAR num data;
...
RMQGetMsgData myrmqmsg, data;
! Handle data
```

Daten vom Datentyp `num` werden aus der Variablen `myrmqmsg` abgerufen und in der Variablen `data` gespeichert.

Argumente

```
RMQGetMsgData Message Data
```

`Message`

Datentyp: `rmqmessage`

Variable mit der empfangenen RMQ-Meldung.

`Data`

Datentyp: `anytype`

Variable vom erwarteten Datentyp, zum Speichern der empfangenen Daten verwendet.

Programmabarbeitung

Mit der Instruktion `RMQGetMsgData` werden die aktuellen Daten in der RMQ-Meldung abgerufen. Konvertieren Sie sie in binäre Daten, kompilieren Sie die Daten, um zu bestimmen, ob sie in der Variablen gespeichert werden können, die in der Instruktion angegeben wird. Kopieren Sie sie dann in die Variable.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_RMQ_VALUE`	Die empfangene Meldung und das Argument `Data` weisen nicht den gleichen Datentyp auf.
`ERR_RMQ_DIM`	Der Datentyp ist gleich, doch die Dimension der Daten in der Meldung und der im Argument `Data` verwendeten Variablen stimmen nicht überein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	Fehlerursache
ERR_RMQ_MSGSIZE	Die Größe der empfangenen Daten überschreitet die für die RMQ der empfangenden Task konfigurierte maximale Größe.
ERR_RMQ_INVMSG	Dieser Fehler wird verworfen, wenn die Meldung ungültig ist. Das könnte beispielsweise passieren, wenn eine PC-Anwendung eine fehlerhafte Meldung sendet.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion RMQGetMsgData sind unten aufgeführt.

Beispiel 1:

```
RECORD mydatatype
  int x;
  int y;
ENDRECORD

VAR intnum msgreceive;
VAR mydatatype mydata;

PROC main()
  ! Setup interrupt
  CONNECT msgreceive WITH msghandler;
  ! Order cyclic interrupt to occur for data type mydatatype
  IRMQMessage mydata, msgreceive;
  WHILE TRUE DO
    ! Performing cycle
    ...
  ENDWHILE
ENDPROC

TRAP msghandler
  VAR rmqmessage message;
  VAR rmqheader header;

  ! Get the RMQ message
  RMQGetMessage message;
  ! Copy RMQ header information
  RMQGetMsgHeader message \Header:=header;

  IF header.datatype = "mydatatype" AND header.ndim = 0 THEN
    ! Copy the data from the message
    RMQGetMsgData message, mydata;
  ELSE
    TPWrite "Received a type not handled or with wrong dimension";
  ENDIF
ENDTRAP
```

Beim Empfang einer neuen Meldung wird die TRAP-Routine msghandler ausgeführt, und die neue Meldung wird in die Variable message kopiert (Instruktion RMQGetMessage). Dann werden die RMQ-Headerdaten kopiert (Instruktion

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.214 RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

`RMQGetMsgHeader`). Wenn die Meldung den erwarteten Datentyp und die richtige Dimension aufweist, werden die Daten in die Variable `mydata` kopiert (Instruktion `RMQGetMsgData`).

Syntax

```
RMQGetMsgData
  [ Message ':=' ] < variable (VAR) of rmqmessage > ','
  [ Data ':=' ] < reference (VAR) of anytype > ';'
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*
`RMQ Message`	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.215 RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung

Verwendung

`RMQGetMsgHeader` (*RAPID Message Queue Get Message Header*) Abrufen der Headerinformationen in der empfangenen RMQ-Meldung und Speichern der Informationen in Variablen vom Typ `rmqheader`, `rmqslot` oder `num`.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `RMQGetMsgHeader`:

Siehe auch *Weitere Beispiele auf Seite 638*.

Beispiel 1:

```
VAR rmqmessage myrmqmsg;
VAR rmqheader myrmqheader;
...
RMQGetMsgHeader myrmqmsg, \Header:=myrmqheader;
```

In diesem Beispiel wird die Variable `myrmqheader` mit Daten aufgefüllt, die aus dem `rmqheader`-Teil der Variablen `myrmqmsg` kopiert wurden.

Beispiel 2:

```
VAR rmqmessage rmqmessage1;
VAR rmqheader rmqheader1;
VAR rmqslot rmqslot1;
VAR num userdef := 0;
...
RRMQGetMsgHeader rmqmessage1 \Header:=rmqheader1 \SenderId:=rmqslot1
     \UserDef:=userdef;
```

In diesem Beispiel werden die Variablen `rmqheader1`, `rmqslot1` und `userdef` mit Daten aufgefüllt, die aus der Variablen `rmqmessage1` kopiert wurden.

Argumente

```
RMQGetMsgHeader Message [\Header] [\SenderId] [\UserDef]
```

`Message`

Datentyp: `rmqmessage`

Variable mit der empfangenen RMQ-Meldung, aus der die Informationen über die Meldung kopiert werden sollen.

`[\Header]`

Datentyp: `rmqheader`

Variable zum Speichern der RMQ-Headerinformationen, die aus der als Parameter `Message` angegebenen Variablen kopiert werden.

`[\SenderId]`

Datentyp: `rmqslot`

Variable zum Speichern der Absenderidentitätsinformationen, die aus der als Parameter `Message` angegebenen Variablen kopiert werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.215 RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung

FlexPendant Interface, PC Interface, or Multitasking
Fortsetzung

[\UserDef]

User Defined data

Datentyp: num

Variable zum Speichern der benutzerdefinierten Daten, die aus der als Parameter Message angegebenen Variablen kopiert werden. Um gültige Daten für diese Variable zu erhalten, muss der Absender angeben, dass diese beim Senden einer RMQ-Meldung eingeschlossen werden sollen . Wenn die Variable nicht verwendet wird, wird der Wert auf -1 gesetzt.

Programmabarbeitung

Mit der Instruktion RMQGetMsgHeader werden die Headerinformationen in der empfangenen RMQ-Meldung abgerufen und je nach den verwendeten Argumenten in Variablen vom Typ rmqheader, rmqslot oder num kopiert.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion RMQGetMsgHeader sind unten aufgeführt.

Beispiel 1:

```
RECORD mydatatype
  int x;
  int y;
ENDRECORD

VAR intnum msgreceive;
VAR mydatatype mydata;

PROC main()
  ! Setup interrupt
  CONNECT msgreceive WITH msghandler;
  ! Order cyclic interrupt to occur for data type mydatatype
  IRMQMessage mydata, msgreceive;
  WHILE TRUE DO
    ! Performing cycle
    ...
  ENDWHILE
ENDPROC

TRAP msghandler
  VAR rmqmessage message;
  VAR rmqheader header;

  ! Get the RMQ message
  RMQGetMessage message;
  ! Copy RMQ header information
  RMQGetMsgHeader message \Header:=header;

  IF header.datatype = "mydatatype" AND header.ndim = 0 THEN
    ! Copy the data from the message
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.215 RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

```
    RMQGetMsgData message, mydata;
  ELSE
    TPWrite "Received a type not handled or with wrong dimension";
  ENDIF
ENDTRAP
```

Beim Empfang einer neuen Meldung wird die TRAP-Routine `msghandler` ausgeführt, und die neue Meldung wird in die Variable `message` kopiert (Instruktion `RMQGetMessage`). Dann werden die RMQ-Headerdaten kopiert (Instruktion `RMQGetMsgHeader`). Wenn die Meldung den erwarteten Datentyp und die richtige Dimension aufweist, werden die Daten in die Variable `mydata` kopiert (Instruktion `RMQGetMsgData`).

Syntax

```
RMQGetMsgHeader
  [ Message ':=' ] < variable (VAR) of rmqmessage > ','
  [ '\' Header ':=' < variable (VAR) of rmqheader >
  [ '\' SenderId ':=' < variable (VAR) of rmqslot >
  [ '\' UserDef ':=' < variable (VAR) of num > ';'
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*
`RMQ Slot`	*rmqslot - Identitätsnummer eines RMQ-Clients auf Seite 1776*
`RMQ Header`	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*
`RMQ Message`	*rmqheader - RAPID Message Queue-Meldungsheader auf Seite 1772*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.216 RMQReadWait – Rückgabe der Meldung von der RMQ

Verwendung

`RMQReadWait` wird im Synchron-Modus verwendet, um alle Meldungsarten zu empfangen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `RMQReadWait`:

Siehe auch *Weitere Beispiele auf Seite 641*.

Beispiel

```
VAR rmqmessage myrmqmsg;
RMQReadWait myrmqmsg;
```

Die erste Meldung in der Warteschlange wird in der Variablen `myrmqmsg` empfangen.

Argumente

```
RMQReadWait Message [\TimeOut]
```

`Message`

Datentyp: `rmqmessage`

Die Variable, in die die empfangene Meldung eingefügt wird.

`[\Timeout]`

Datentyp: `num`

Die maximale Zeitdauer [s], für die die Programmabarbeitung auf eine Meldung wartet. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_RMQ_TIMEOUT` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt. Es ist möglich, den Timeout auf 0 (null) Sekunden zu setzen, damit keine Wartezeit anfällt.

Wenn der Parameter `\Timeout` nicht verwendet wird, beträgt die Wartezeit 60 s. Um für eine unendliche Dauer zu warten, verwenden Sie die vordefinierte Konstante `WAIT_MAX`.

Programmabarbeitung

Alle eingehenden Meldungen kommen in die Warteschleife und `RMQReadWait` arbeitet die Meldungen eine nach der anderen in der Reihenfolge ihres Auftretens (FIFO) ab. Es liegt am Benutzer, eine volle Warteschlange zu vermeiden und jede von der RAPID Message Queue unterstützte Meldungsart zu verarbeiten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_RMQ_TIMEOUT`	Innerhalb der Timeout-Zeit wurde keine Antwort empfangen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	Fehlerursache
ERR_RMQ_INVMSG	Dieser Fehler wird verworfen, wenn die Meldung ungültig ist. Das könnte beispielsweise passieren, wenn eine PC-Anwendung eine fehlerhafte Meldung sendet.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion RMQReadWait sind unten aufgeführt.

Beispiel 1:

```
VAR rmqmessage myrmqmsg;
RMQReadWait myrmqmsg \TimeOut:=30;
```

Die erste Meldung in der Warteschlange wird in der Variablen myrmqmsg empfangen. Geht innerhalb von 30 Sekunden keine Meldung ein, wird die Abarbeitung des Programms gestoppt.

Beispiel 2:

```
PROC main()
  VAR rmqmessage myrmqmsg;
  FOR i FROM 1 TO 25 DO
    RMQReadWait myrmqmsg \TimeOut:=30;
    ...
  ENDFOR

ERROR
  IF ERRNO = ERR_RMQ_TIMEOUT THEN
    TPWrite "ERR_RMQ_TIMEOUT error reported";
    ...
  ENDIF
ENDPROC
```

Meldungen werden von der Warteschlange empfangen und in der Variablen myrmqmsg gespeichert. Wenn der Empfang einer Meldung länger als 30 Sekunden dauert, wird die Fehlerbehandlung aufgerufen.

Einschränkungen

RMQReadWait wird nur im Synchron-Modus unterstützt. Die Abarbeitung dieser Instruktion im interruptbasierten Modus führt zu einem schwerwiegenden Laufzeitfehler.

RMQReadWait wird weder auf Interrupt- noch auf Benutzerebene unterstützt. Die Abarbeitung dieser Instruktion auf einer dieser Ebenen führt zu einem schwerwiegenden Laufzeitfehler.

Syntax

```
RMQReadWait
  [ Message ':=' ] < variable (VAR) of rmqmessage>
  [ '\' TimeOut':=' < expression (IN) of num > ] ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.216 RMQReadWait – Rückgabe der Meldung von der RMQ

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Beschreibung der Task-Modi	*Technisches Referenzhandbuch - Systemparameter*
`rmqmessage` Datentyp	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*.
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*.
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*.
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*.
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*.
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*.
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*.
Leere RAPID Message Queue	*RMQEmptyQueue - Leere RAPID Message Queue auf Seite 627*.
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.217 RMQSendMessage - Senden einer RMQ-Datenmeldung

Verwendung

`RMQSendMessage` (*RAPID Message Queue Send Message*) wird verwendet, um Daten an eine für eine RAPID-Task konfigurierte RMQ oder einen Robot Application Builder-Client zu senden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `RMQSendMessage`:

Siehe auch *Weitere Beispiele auf Seite 644*.

Beispiel 1:

```
VAR rmqslot destination_slot;
VAR string data:="Hello world";
..
RMQFindSlot destination_slot,"RMQ_Task2";
RMQSendMessage destination_slot,data;
```

In dem Beispiel wird gezeigt, wie der Wert in der Variablen `data` an die RAPID-Task `"Task2"` mit der konfigurierten RMQ "`RMQ_Task2`" gesendet wird.

Beispiel 2:

```
VAR rmqslot destination_slot;
CONST robtarget p5:=[ [600, 500, 225.3], [1, 0, 0, 0], [1, 1, 0,
     0], [ 11, 12.3, 9E9, 9E9, 9E9, 9E9] ];
VAR num my_id:=1;
..
RMQFindSlot destination_slot,"RMQ_Task2";
RMQSendMessage destination_slot, p5 \UserDef:=my_id;
my_id:=my_id + 1;
```

In dem Beispiel wird gezeigt, wie der Wert in der Konstanten `p5` an die RAPID-Task `"Task2"` mit der konfigurierten RMQ "RMQ_Task2" gesendet wird. Es wird außerdem eine benutzerdefinierte Zahl gesendet. Diese Zahl kann vom Empfänger als Bezeichner verwendet werden.

Argumente

```
RMQSendMessage Slot SendData [\UserDef]
```

`Slot`

Datentyp: `rmqslot`

Die Slot-Identitätsnummer des Clients, der die Meldung empfangen soll.

`SendData`

Datentyp: `anytype`

Verweis auf eine Variable, Persistente oder Konstante mit den Daten, die an den Client mit der im Argument `Slot` angegebenen Identität gesendet werden sollen.

`[\UserDef]`

User Defined data

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.217 RMQSendMessage - Senden einer RMQ-Datenmeldung
FlexPendant Interface, PC Interface, or Multitasking
Fortsetzung

Daten mit benutzerdefinierten Informationen für den Empfänger von `SendData`, d. h. für den Client mit der in der Variablen `Slot` angegebenen Identitätsnummer. Der Wert muss eine Ganzzahl zwischen 0 und 32767 sein.

Programmabarbeitung

Mit der Instruktion `RMQSendMessage` werden Daten an einen angegebenen Client gesendet. Mit der Instruktion werden die Eingangsdaten in einen Speichercontainer gepackt und gesendet.

Wenn der empfangende Client keine Meldungen empfangen möchte, d. h., wenn er keinen Interrupt für den in der `RMQSendMessage`-Instruktion angegebenen Datentyp eingerichtet hat oder nicht auf eine `RMQSendWait`-Instruktion wartet, wird die Meldung verworfen und es wird eine Warnung generiert.

Mit der Instruktion können nicht alle Datentypen gesendet werden (siehe Einschränkungen).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_RMQ_MSGSIZE`	Die Meldung ist zu groß. Entweder überschreiten die Daten die maximal zulässige Meldungsgröße oder der empfangende Client ist nicht zum Empfangen von Daten mit der gesendeten Größe konfiguriert.
`ERR_RMQ_FULL`	Die Meldungswarteschlange des Ziels ist voll.
`ERR_RMQ_INVALID`	Der Ziel-Slot wurde nicht verbunden oder ist nicht mehr verfügbar. Wenn keine Verbindung besteht, muss `RMQFindSlot` aufgerufen werden. Wenn der Slot nicht verfügbar ist, wurde die Verbindung eines Remote-Clients mit der Steuerung getrennt.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `RMQSendMessage` sind unten aufgeführt.

Beispiel 1:

```
MODULE SenderMod
  RECORD msgrec
    num x;
    num y;
  ENDRECORD

  PROC main()
    VAR rmqslot destinationSlot;
    VAR msgrec msg :=[0, 0, 0];

    ! Connect to a Robot Application Builder client
    RMQFindSlot destinationSlot ”My_RAB_client”;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
    ! Perform cycle
    WHILE TRUE DO
      ! Update msg with valid data
      ...
      ! Send message
      RMQSendMessage destinationSlot, msg;
      ...
    ENDWHILE
  ERROR
    IF ERRNO = ERR_RMQ_INVALID THEN
      ! Handle destination client lost
      WaitTime 1;
      ! Reconnect to Robot Application Builder client
      RMQFindSlot destinationSlot ”My_RAB_client”;
      ! Avoid execution stop due to retry count exceed
      ResetRetryCount;
      RETRY;
    ELSIF ERRNO = ERR_RMQ_FULL THEN
      ! Handle destination queue full
      WaitTime 1;
      ! Avoid execution stop due to retry count exceed
      ResetRetryCount;
      RETRY;
    ENDIF
  ENDPROC
ENDMODULE
```

Das Beispiel veranschaulicht die Verwendung der Instruktion `RMQSendMessage` ohne die Behandlung von Laufzeitfehlern. Das Programm sendet benutzerdefinierte Daten vom Typ `msgrec` an den Robot Application Builder-Client `My_RAB_client`.

Einschränkungen

Das Einrichten von Interrupts und das Senden oder Empfangen von Dateninstanzen von Datentypen mit keinem Wert (Non-Value), festem Wert (Semi-Value) oder vom Datentyp `motsetdata` ist nicht möglich.

Die maximale Datenmenge, die an einen Robot Application Builder-Client gesendet werden kann, beträgt ca. 5000 Byte. Die maximale Datenmenge, die von einer RMQ empfangen und in einer `rmqmessage` gespeichert werden kann, beträgt ca. 3000 Byte. Die Größe der Daten, die von einer RMQ empfangen werden kann, lässt sich konfigurieren (Standardgröße 400, maximale Größe 3000).

Syntax

```
RMQSendMessage
  [ Slot ':=' ] < variable (VAR) of rmqslot > ','
  [ SendData ':=' ] < reference (REF) of anytype >
  [ '\' UserDef ':=' < expression (IN) of num > ] ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.217 RMQSendMessage - Senden einer RMQ-Datenmeldung

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange einer RAPID-Task und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*
`RMQ Slot`	*rmqslot - Identitätsnummer eines RMQ-Clients auf Seite 1776*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.218 RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort

Verwendung

Die Instruktion `RMQSendWait` (*RAPID Message Queue Send Wait*) ermöglicht das Senden von Daten an eine RMQ oder an einen Robot Application Builder-Client und das Warten auf eine Antwort vom angegebenen Client. Bei Verwendung dieser Instruktion muss der Benutzer wissen, welcher Datentyp in der Antwort vom Client gesendet wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `RMQSendWait`:

Siehe auch *Weitere Beispiele auf Seite 650*.

Beispiel 1:

```
VAR rmqslot destination_slot;
VAR string sendstr:="This string is from T_ROB1";
VAR rmqmessage receivemsg;
VAR num mynum;
..
RMQFindSlot destination_slot, "RMQ_T_ROB2";
RMQSendWait destination_slot, sendstr, receivemsg, mynum;
RMQGetMsgData receivemsg, mynum;
```

In dem Beispiel wird gezeigt, wie die Daten in der Variablen `sendstr` an die RAPID-Task `T_ROB2` mit der konfigurierten RMQ RMQ "`RMQ_T_ROB2` gesendet werden. Jetzt wartet die Instruktion `RMQSendWait` auf eine Antwort von der Task T_ROB2. Die Instruktion in T_ROB2 muss Daten vom Datentyp `num` senden, um die Warteinstruktion `RMQSendWait` zu beenden. Wenn die Meldung empfangen wurde, werden die Daten mit der Instruktion `RMQGetMsgData` aus der Variablen `receivemsg` in die Variable `mynum` kopiert

Beispiel 2:

```
VAR rmqslot rmqslot1;
VAR string mysendstr;
VAR rmqmessage rmqmessage1;
VAR string receivestr;
VAR num mysendid:=1;
..
mysendstr:="Message from Task1";
RMQFindSlot rmqslot1, "RMQ_Task2";
RMQSendWait rmqslot1, mysendstr \UserDef:=mysendid, rmqmessage1,
     receivestr \TimeOut:=20;
RMQGetMsgData rmqmessage1, receivestr;
mysendid:=mysendid + 1;
```

In dem Beispiel wird gezeigt, wie die Daten in der Variablen `mysendstr` an die RAPID-Task "`Task2`" mit der konfigurierten RMQ"`RMQ_Task2`" gesendet werden. Es wird außerdem eine benutzerdefinierte Zahl gesendet. Diese Zahl kann vom Empfänger als Bezeichner verwendet werden. Sie muss an den Empfänger zurückgesendet werden, um die Warteinstruktion `RMQSendWait` zu beenden. Zum

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.218 RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort
FlexPendant Interface, PC Interface, or Multitasking
Fortsetzung

Beenden der Warteinstruktion muss außerdem vom Client der richtige Datentyp gesendet werden. Dieser Datentyp wird von der Variablen `receivestr` in der `RMQSendWait`-Instruktion angegeben. Nachdem die Meldung empfangen wurde, werden die aktuellen Daten mit der Instruktion `RMQGetMsgData` in die Variable `receivestr` kopiert

Argumente

```
RMQSendWait Slot SendData [\UserDef] Message ReceiveDataType
    [\TimeOut]
```

`Slot`

Datentyp: `rmqslot`

Die Identitätsnummer des Clients, der die Meldung empfangen soll.

`SendData`

Datentyp: `anytype`

Verweis auf eine Variable, Persistente oder Konstante mit den Daten, die an den Client mit der in der Variablen `Slot` angegebenen Identitätsnummer gesendet werden sollen.

`[\UserDef]`

User Defined data

Datentyp: `num`

Daten mit benutzerdefinierten Informationen für den Empfänger von `SendData`, d. h. für den Client mit der in der Variablen `Slot` angegebenen Identitätsnummer. Bei Verwendung dieses optionalen Arguments wird die `RMQSendWait`-Instruktion nur beendet, wenn der `ReceiveDataType` und das angegebene `UserDef` den Angaben in der Meldungsantwort entsprechen. Der Wert muss eine Ganzzahl zwischen 0 und 32767 sein.

`Message`

Datentyp: `rmqmessage`

Die Variable, in die die empfangene Meldung eingefügt wird.

`ReceiveDataType`

Datentyp: `anytype`

Ein Verweis auf eine Persistente, Variable oder Konstante des Datentyps, auf den die Instruktion wartet. Die eigentlichen Daten werden nicht in diese Variable kopiert, wenn `RMQSendWait` ausgeführt wird. Mit diesem Argument wird lediglich der Datentyp angegeben, auf den die `RMQSendWait`-Instruktion wartet.

`[\Timeout]`

Datentyp: `num`

Die maximale Zeitdauer [s], für die die Programmabarbeitung auf eine Antwort wartet. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_RMQ_TIMEOUT` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn der Parameter \Timeout nicht verwendet wird, beträgt die Wartezeit 60 s. Um für eine unendliche Dauer zu warten, verwenden Sie die vordefinierte Konstante WAIT_MAX.

Programmabarbeitung

Die Instruktion RMQSendWait sendet Daten und wartet auf eine Antwort von dem Client mit der angegebenen Slot-Identität. Die Antwort muss eine rmqmessage von dem Client sein, der die Meldung empfangen hat, und die Antwort muss von dem Datentyp sein, der im Argument ReceiveDataType angegeben wird. Die Meldung wird auf die gleiche Weise wie bei Verwendung von RMQSendMessage gesendet, d. h. der Empfänger erhält eine normale RAPID Message Queue-Meldung. Der Absender muss sicherstellen, dass der Empfänger weiß, dass eine Antwort erforderlich ist. Wenn in RMQSendWait das optionale Argument UserDef verwendet wird, muss der empfangende Client in der Antwort dasselbe UserDef verwenden

Wenn der empfangende Client keine Meldungen empfangen möchte, d. h., wenn er keinen Interrupt für den in der RMQSendWait-Instruktion angegebenen Datentyp eingerichtet hat, wird die Meldung verworfen und es wird eine Warnung generiert. Die Instruktion gibt nach der im Argument TimeOut angegebenen Zeitspanne oder nach dem Standard-Timeout von 60 s einen Fehler zurück. Dieser Fehler kann in einer Fehlerbehandlungsroutine behandelt werden.

Die RMQSendWait-Instruktion hat die höchste Priorität, wenn eine Meldung empfangen wird, die der Beschreibung der erwarteten Antwort und der Meldung entspricht, die mit einer TRAP-Routine verbunden ist (siehe Instruktion IRMQMessage).

Wenn beim Warten auf eine Antwort vom Client ein Stromausfall auftritt, wird die Variable im Argument Slot auf 0 gesetzt und die Instruktion wird erneut abgearbeitet. Die Instruktion schlägt aufgrund einer ungültigen Slot-Identität fehl, weshalb die Fehlerbearbeitung mit dem Fehlercode ERR_RMQ_INVALID aufgerufen wird, vorausgesetzt, es wurde eine definiert. Dort kann die Slot-Identität neu initialisiert werden.

Mit der Instruktion können nicht alle Datentypen gesendet werden (siehe Einschränkungen).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_RMQ_MSGSIZE	Die Meldung ist zu groß. Entweder überschreiten die Daten die maximal zulässige Meldungsgröße oder der empfangende Client ist nicht zum Empfangen von Daten mit der gesendeten Größe konfiguriert.
ERR_RMQ_FULL	Die Meldungswarteschlange des Ziels ist voll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.218 RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort
FlexPendant Interface, PC Interface, or Multitasking
Fortsetzung

Name	Fehlerursache
ERR_RMQ_INVALID	Der rmqslot wurde nicht initialisiert oder der Ziel-Slot ist nicht mehr verfügbar. Dieser Fall kann eintreten, wenn der Ziel-Slot ein Remote-Client ist und dieser nicht mehr mit der Steuerung verbunden ist. Oder RMQSendWait wurde durch einen Stromausfall unterbrochen, und beim Neustart wird rmqslot auf 0 gesetzt.
ERR_RMQ_TIMEOUT	Innerhalb der Timeout-Zeit wurde keine Antwort empfangen.
ERR_RMQ_INVMSG	Dieser Fehler wird verworfen, wenn die Meldung ungültig ist. Das könnte beispielsweise passieren, wenn eine PC-Anwendung eine fehlerhafte Meldung sendet.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion RMQSendWait sind unten aufgeführt.

Beispiel 1:

```
MODULE RMQ_Task1_mod
  PROC main()
    VAR rmqslot destination_slot;
    VAR string mysendstr:="String sent from RMQ_Task1_mod";
    VAR string myrecstr;
    VAR rmqmessage recmsg;
    VAR rmqheader header;

    !Get slot identity to client called RMQ_Task2
    RMQFindSlot destination_slot, "RMQ_Task2";

    WHILE TRUE DO
      ! Do something
      ...
      !Send data in mysendstr, wait for an answer of type string
      RMQSendWait destination_slot, mysendstr, recmsg, myrecstr;
      !Get information about the received message
      RMQGetMsgHeader recmsg \Header:=header;
      IF header.datatype = "string" AND header.ndim = 0 THEN
        ! Copy the data in recmsg
        RMQGetMsgData recmsg, myrecstr;
        TPWrite "Received string: " + myrecstr;
      ELSE
        TPWrite "Not a string that was received";
      ENDIF
    ENDWHILE
  ENDPROC
ENDMODULE
```

Die Daten in der Variablen mysendstr werden mit der Instruktion RMQSendWait und der konfigurierten RAPID Message Queue RMQ_Task2 an die RAPID-Task Task2 gesendet. Die Antwort von der RAPID-Task Task2 muss eine Zeichenfolge (als Datentyp in der Variablen myrecstr angegeben) sein. Die RMQ-Meldung wird in der Variablen recmsg empfangen. Die Variable myrecstr wird im Aufruf von

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`RMQSendWait` lediglich verwendet, um den Datentyp anzugeben, den der Absender als Antwort erwartet. Die Variable im Aufruf von `RMQSendWait` enthält keine gültigen Daten

Einschränkungen

Es ist nicht erlaubt, `RMQSendWait` im Synchron-Modus abzuarbeiten. Dies würde zu einem schwerwiegenden Laufzeitfehler führen.

Das Einrichten von Interrupts und das Senden oder Empfangen von Dateninstanzen von Datentypen mit keinem Wert (Non-Value), festem Wert (Semi-Value) oder vom Datentyp `motsetdata` ist nicht möglich.

Die maximale Datenmenge, die an einen Robot Application Builder-Client gesendet werden kann, beträgt ca. 5000 Byte. Die maximale Datenmenge, die von einer RMQ empfangen und in einer `rmqmessage` gespeichert werden kann, beträgt ca. 3000 Byte. Die Größe der Daten, die von einer RMQ empfangen werden kann, lässt sich konfigurieren (Standardgröße 400, maximale Größe 3000).

Syntax

```
RMQSendWait
  [ Slot ':=' ] < variable (VAR) of rmqslot > ','
  [ SendData ':=' ] < reference (REF) of anytype >
  [ '\' UserDef ':=' < expression (IN) of num > ] ','
  [ Message' :=' ] < variable (VAR) of rmqmessage > ','
  [ ReceiveDataType ':=' ] < reference (REF) of anytype > ','
  [ '\' Timeout ':=' < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer eines RAPID-Message Queue-Task	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Senden von Daten an die Warteschlange eines RAPID-Task	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*
`RMQ Slot`	*rmqslot - Identitätsnummer eines RMQ-Clients auf Seite 1776*
`RMQ Message`	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.219 SafetyControllerSyncRequest - Initiierung des Hardware-Synchronisationsverfahrens

Verwendung

`SafetyControllerSyncRequest` wird verwendet, um das Hardware-Synchronisationsverfahren einzuleiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SafetyControllerSyncRequest`.

Beispiel 1:

```
SafetyControllerSyncRequest;
```

Hardware-Synchronisationsverfahren einleiten.

Programmabarbeitung

Diese Instruktion muss vor der Aktivierung des Synchronisationssignals aufgerufen werden.

Syntax

```
SafetyControllerSyncRequest ';'
```

Weitere Informationen

Information über	Siehe
`SafetyControllerGetChecksum`	*SafetyControllerGetChecksum - Abrufen der Prüfsumme der Benutzerkonfigurationsdatei auf Seite 1490*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung auf Seite 1492*
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Prüfsumme für geschützte Parameter abrufen auf Seite 1493*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.220 Save - Speichern eines Programmmoduls

Verwendung

`Save` wird verwendet, um ein Programmmodul zu speichern.

Das angegebene Programmmodul im Programmspeicher wird mit dem ursprünglichen (in `Load` oder `StartLoad`) oder dem angegebenen Dateipfad gespeichert.

Es ist auch möglich, ein Systemmodul im angegebenen Dateipfad zu speichern.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Save`:

Siehe auch *Weitere Beispiele auf Seite 654*.

Beispiel 1:

```
Load "HOME:/PART_B.MOD";
...
Save "PART_B";
```

Lädt das Programmmodul mit dem Dateinamen `PART_B.MOD` aus `HOME:` in den Programmspeicher.

Speichert das Programmmodul `PART_B` mit dem ursprünglichen Dateipfad `HOME:` und dem ursprünglichen Dateinamen `PART_B.MOD`.

Argumente

```
Save [\TaskRef]|[\TaskName] ModuleName [\FilePath] [\File]
```

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, in der das Programmmodul gespeichert werden soll.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“ (die Variablenidentität für die Task `T_ROB1` lautet z. B. `T_ROB1Id`).

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, in der das Programmmodul gespeichert werden soll.

Wenn keines der Argumente `\TaskRef` oder `\TaskName` angegeben ist, wird das angegebene Programmmodul in der aktuellen (ausgeführten) Programmtask gespeichert.

`ModuleName`

Datentyp: `string`

Das zu speichernde Programmmodul.

`[\FilePath]`

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.220 Save - Speichern eines Programmmoduls
RobotWare - OS
Fortsetzung

Pfad und Name der Datei, unter dem das Programmmodul gespeichert werden soll. Der Dateiname soll ausgeschlossen werden, wenn das Argument `\File` verwendet wird.

`[\File]`

Datentyp: `string`

Wenn der Dateiname im Argument `\FilePath` ausgelassen wird, muss er mit diesem Argument angegeben werden.

Das Argument `\FilePath\File` kann nur für Programmmodule ausgelassen werden, die mit `Load` oder `StartLoad-WaitLoad` geladen wurden. Das Programmmodul wird dann an dem Ort gespeichert, der in diesen Instruktionen angegeben ist. Um das Programmmodul an einem anderen Ort zu speichern, verwenden Sie das Argument `\FilePath \File`.

Um ein zuvor vom FlexPendant, einem externen Computer oder der Systemkonfiguration geladenes Programmmodul zu speichern, ist das Argument `\FilePath \File` erforderlich.

Programmabarbeitung

Die Programmabarbeitung wartet das Ende des Speichervorgangs ab, bevor sie mit der nächsten Instruktion fortfährt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_IOERROR`	Die zu speichernde Datei kann nicht geöffnet werden, weil der Zugriff nicht erlaubt, das Verzeichnis nicht vorhanden oder kein Speicherplatz mehr auf dem Medium vorhanden ist.
`ERR_MODULE`	Das Programmmodul kann nicht gespeichert werden, weil ein unbekannter, unklarer oder gar kein Modulname angegeben ist.
`ERR_PATH`	Das Argument `\FilePath` ist nicht für Programmmodule angegeben, die vom FlexPendant, über die Systemparameter oder von einem externen Computer geladen wurden.
`ERR_TASKNAME`	Der Name der Programmtask im Argument `\TaskName` kann im System nicht gefunden werden.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `Save` sind unten aufgeführt.

Beispiel 1:

```
Save "PART_A" \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

Speichert das Programmmodul `PART_A` in `HOME:` in der Datei `PART_A.MOD` und im Verzeichnis `DOORDIR`.

Beispiel 2:

```
Save "PART_A" \FilePath:="HOME:" \File:="DOORDIR/PART_A.MOD";
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wie in obigem Beispiel 1, aber andere Syntax.

Beispiel 3

```
Save \TaskRef:=TSK1Id, "PART_A"
     \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

Speichert das Programmmodul `PART_A` in Programmtask `TSK1` am angegebenen Speicherort. In diesem Beispiel wird die Instruktion `Save` in einer Programmtask abgearbeitet und das Speichern erfolgt in einer anderen Programmtask.

Beispiel 4:

```
Save \TaskName:="TSK1", "PART_A"
     \FilePath:="HOME:/DOORDIR/PART_A.MOD";
```

Speichert das Programmmodul `PART_A` in der Programmtask `TSK1` am angegebenen Speicherort. Dies ist ein weiteres Beispiel, in dem die Instruktion `Save` in einer Programmtask abgearbeitet und in einer anderen Programmtask gespeichert wird.

Einschränkungen

Interrupt-Routinen, E/A-Systemereignisse und andere Programmtasks können während des Speichervorgangs nicht abgearbeitet werden. Daher werden alle derartigen Vorgänge verzögert.

Der Speichervorgang kann die Aktualisierung von PERS-Daten unterbrechen, die schrittweise aus anderen Programmtasks erfolgt. Dies führt zu inkonsistenten PERS-Daten.

Ein Programmstopp während der Abarbeitung der Save-Instruktion kann zu einem Sicherheitsstopp mit MOTORS OFF führen. Auf dem FlexPendant wird die Fehlermeldung „20025 Stop Anweisung Timeout“ angezeigt.

Verhindern Sie Roboterbewegungen während des Speichervorgangs.

Syntax

```
Save
  [[ '\' TaskRef ':=' <variable (VAR) of taskid>]
  |[ '\' TaskName' :=' <expression (IN) of string>] ',']
  [ ModuleName' :=' ] <expression (IN) of string>
  [ '\' FilePath' :='<expression (IN) of string> ]
  [ '\' File' :=' <expression (IN) of string>] ';'
```

Weitere Informationen

Information über	Siehe
Programmieren von Tasks	*taskid – Taskidentifizierung auf Seite 1818*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.221 SaveCfgData - Speichert Systemparameter in einer Datei

Verwendung

`SaveCfgData` dient dazu, Systemparameter in einer Datei zu speichern. Dies kann nützlich sein, nachdem die Systemparameter mit der Instruktion `WriteCfgData` aktualisiert wurden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SaveCfgData`.

Beispiel 1:

```
SaveCfgData "SYSPAR" \File:="MYEIO.cfg", EIO_DOMAIN;
```

Speichert die E/A-Konfigurationsdomain in der Datei `MYEIO.cfg` im Verzeichnis `SYSPAR`.

Beispiel 2:

```
SaveCfgData "SYSPAR", ALL_DOMAINS;
```

Speichert alle vorhandenen Konfigurationsdomains im Verzeichnis `SYSPAR`. Die Dateien erhalten die Namen `EIO.cfg`, `MMC.cfg`, `PROC.cfg`, `SIO.cfg`, `SYS.cfg` und `MOC.cfg`.

Argumente

```
SaveCfgData FilePath [\File] Domain
```

`FilePath`

Datentyp: `string`

Pfad und Name der Datei, unter denen die Datei gespeichert werden soll. Der Dateiname soll ausgeschlossen werden, wenn das Argument `\File` verwendet wird.

`[\File]`

Datentyp: `string`

Wenn der Dateiname im Argument `\FilePath` ausgelassen wird, muss er mit diesem Argument angegeben werden.

`Domain`

Datentyp: `cfgdomain`

Die zu speichernde Domain des Systemparameters.

Programmabarbeitung

Speichert Systemparameter in einer Datei.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_CFG_ILL_DOMAIN	Die verwendete cfgdomain ist ungültig oder wird nicht verwendet.
ERR_CFG_WRITEFILE	Das Verzeichnis existiert nicht, oder der verwendete FilePath und die verwendete File sind ein Verzeichnis, oder es gibt ein anderes Problem beim Speichern der Datei.

Syntax

```
SaveCfgData
  [FilePath ':=' ] <expression (IN) of string>
  ['\' File ':=' <expression (IN) of string>]
  [Domain ':=' ] <expression (IN) of cfgdomain> ';'
```

Weitere Informationen

Information über	Siehe
cfgdomain-Daten	*cfgdomain - Konfigurationsdomain auf Seite 1675*
Systemparameter	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.222 SCWrite - Senden von Variablendaten an eine Client-Anwendung

Verwendung

`SCWrite` (*Superior Computer Write*) wird zum Senden des Namens, des Typs, der Dimension und des Werts einer persistenten Variablen an eine Client-Anwendung verwendet. Es können sowohl einzelne Variablen als auch Datenfelder von Variablen gesendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SCWrite`:

Beispiel 1:

```
PERS num cycle_done;

PERS num numarr{2}:=[1,2];

SCWrite cycle_done;
```

Der Name, der Typ, die Dimension und der Wert der persistenten Variablen `cycle_done` werden an alle Client-Anwendungen gesendet.

Beispiel 2:

```
SCWrite \ToNode := "138.221.228.4", cycle_done;
```

Der Name, der Typ, die Dimension und der Wert der persistenten Variablen `cycle_done` werden an alle Client-Anwendungen gesendet. Das Argument `\ToNode` wird ignoriert.

Beispiel 3

```
SCWrite numarr;
```

Der Name, der Typ, die Dimension und der Wert der persistenten Variablen `numarr` werden an alle Client-Anwendungen gesendet.

Beispiel 4:

```
SCWrite \ToNode := "138.221.228.4", numarr;
```

Der Name, der Typ, die Dimension und der Wert der persistenten Variablen `numarr` werden an alle Client-Anwendungen gesendet. Das Argument `\ToNode` wird ignoriert.

Argumente

```
SCWrite [ \ToNode ] Variable
```

`[\ToNode]`

Datentyp: `datatype`

Das Argument wird ignoriert.

`Variable`

Datentyp: `anytype`

Der Name einer persistenten Variablen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Der Name, der Typ, die Dimension und der Wert der persistenten Variablen werden an alle Client-Anwendungen gesendet. Die Dimension „dim“ der Variablen wird nur gesendet, wenn die Variable ein Datenfeld ist.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SC_WRITE	Fehler beim Senden an einen externen Computer.

Die SCWrite-Instruktion gibt in folgenden Fällen eine Fehlermeldung aus:

Die Variable konnte nicht an den Client gesendet werden. Das kann folgende Gründe haben:

- Die SCWrite-Meldungen kommen in so kurzen Abständen, dass sie nicht an den Client gesendet werden können. Lösung: Fügen Sie eine WaitTime-Instruktion zwischen den SCWrite-Instruktionen ein.
- Der Variablenwert ist zu hoch und verkleinert den ARRAY oder RECORD.
- Die Fehlermeldung lautet: `41473 System access error, Failed to send variable arg1`, wobei `arg1` der Name der Variable ist.

Die SCWrite-Instruktion gibt keine Fehlermeldung zurück, wenn die Client-Anwendung z. B. ausgeschaltet wird oder die Kommunikation unterbrochen ist. Das Programm fährt mit der Abarbeitung fort.

SCWrite-Fehlerbehandlung

Um das Anhalten des Programms zu vermeiden, wenn ein Fehler in einer SCWrite -Instruktion auftritt, muss der Fehler mit einer *Fehlerbehandlungsroutine* behandelt werden. Der Fehler wird lediglich an das Protokoll weitergegeben und die Programmabarbeitung wird fortgesetzt.

Denken Sie daran, dass es durch eine Fehlerbehandlungsroutine schwieriger wird, Fehler in der Client-Kommunikation zu finden, da der Fehler niemals auf dem Display des FlexPendant angezeigt wird. (Er kann jedoch im Protokoll gesehen werden.)

Fortsetzung auf nächster Seite

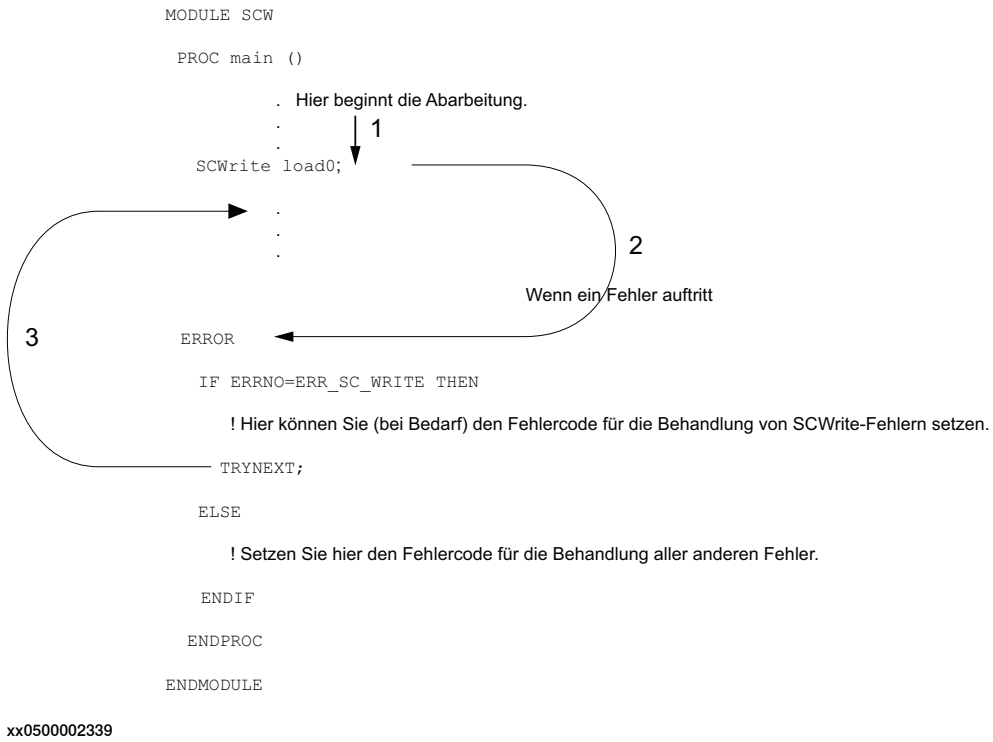
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.222 SCWrite - Senden von Variablendaten an eine Client-Anwendung

PC interface/backup

Fortsetzung

Das RAPID-Programm sieht folgendermaßen aus:



xx0500002339

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.223 SearchC - Kreisförmige Suchbewegung des Roboters

Verwendung

`SearchC` (*Search Circular*) wird verwendet, um mit einer kreisförmigen Bewegung des Werkzeugarbeitspunkts (TCP) eine Position zu suchen.

Während der Bewegung überwacht der Roboter ein digitales Eingangssignal oder eine persistente Variable. Wenn das Signal oder die persistente Variable den erforderlichen Wert annimmt, liest der Roboter sofort die aktuelle Position ab.

Diese Instruktion eignet sich typischerweise für den Einsatz eines Sensors zur Oberflächenerkennung. Mit der Instruktion `SearchC` kann der Umriss eines Werkobjekts ermittelt werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Bei Verwendung von Suchinstruktionen ist es wichtig, das E/A-System so zu konfigurieren, dass zwischen dem Setzen des physischen Signals und dem Ankommen der Informationen über das Setzen im System nur eine sehr kurze Zeitspanne liegt. (Verwenden Sie die E/A-Gerät mit Interruptsteuerung, nicht mit Abfragesteuerung.) Die Vorgehensweise kann von Feldbus zu Feldbus unterschiedlich sein. Bei der Verwendung von DeviceNet sind in den ABB-Einheiten (lokale E/A) kurze Intervalle vorgesehen, da sie den Verbindungstyp *Statuswechsel* verwenden. Achten Sie bei der Verwendung anderer Feldbusse darauf, dass das Netzwerk ordnungsgemäß konfiguriert ist, sodass die richtigen Bedingungen vorliegen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SearchC`:

Siehe auch *Weitere Beispiele auf Seite 668*.

Beispiel 1:

```
SearchC di1, sp, cirpoint, p10, v100, probe;
```

Der TCP der `probe` wird mit der Geschwindigkeit `v100` kreisförmig an die Position `p10` bewegt. Wenn der Wert von Signal `di1` zu aktiv wechselt, wird die Position in `sp` gespeichert.

Beispiel 2:

```
SearchC \Stop, di2, sp, cirpoint, p10, v100, probe;
```

Der TCP der `probe` wird kreisförmig an die Position `p10` bewegt. Wenn der Wert von Signal `di2` zu aktiv wechselt, wird die Position in `sp` gespeichert und der Roboter stoppt sofort.

Beispiel 3

```
PERS bool mypers:=FALSE;
...
SearchC \Stop, mypers, sp, cirpoint, p10, v100, probe;
```

Der TCP der `probe` wird kreisförmig an die Position `p10` bewegt. Wenn der Wert von der persistenten Variable `mypers` zu TRUE wechselt, wird die Position in `sp` gespeichert und der Roboter stoppt sofort.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.223 SearchC - Kreisförmige Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

Argumente

```
SearchC [\Stop] | [\PStop] | [\SStop] | [\Sup] Signal | PersBool
     [\Flanks] | [\PosFlank] | [\NegFlank] | [\HighLevel] |
     [\LowLevel] SearchPoint CirPoint ToPoint [\ID] Speed [\V] |
     [\T] Tool [\WObj] [\Corr] [\TLoad]
```

`[ \Stop ]`

Stiff Stop

Datentyp: `switch`

Die Roboterbewegung stoppt so schnell wie möglich, ohne den TCP auf der Bahn zu halten (harter Stopp), wenn der Wert des Suchsignals zu „aktiv“ oder der Wert der persistenten Variable zu TRUE wechselt. Der Roboter legt jedoch vor dem Stopp eine kurze Strecke zurück und wird nicht an die gesuchte Position zurückbewegt, d. h. an die Position, wo sich das Signal oder der persistente Wert geändert hat.

Testen Sie zunächst eine langsame Geschwindigkeit empfohlen, <100 Beispiel mm/s, die dann schrittweise auf den gewünschten Wert erhöht wird.

WARNUNG

Ein Suchstopp durch einen abrupten Stopp (Schalter `\Stop`) darf nur bei einer TCP-Geschwindigkeit von weniger als 100 mm/s durchgeführt werden. Bei einem abrupten Stopp bei höherer Geschwindigkeit können sich die Achsen möglicherweise unkontrolliert bewegen.

Hinweis

Für einen YuMi-Roboter beträgt die Höchstgeschwindigkeit für Suchen mit abrupten Stopp 1000 mm/s.

`[ \PStop ]`

Path Stop

Datentyp: `switch`

Die Roboterbewegung stoppt so schnell wie möglich, wobei der TCP auf der Bahn gehalten wird (Softstopp), wenn der Wert des Suchsignals zu „aktiv“ oder der Wert der persistenten Variable zu TRUE wechselt. Der Roboter legt jedoch vor dem Stopp eine Strecke zurück und wird nicht an die gesuchte Position zurückbewegt, d. h. an die Position, wo sich das Signal oder der persistente Wert geändert hat.

`[ \SStop ]`

Soft Stop

Datentyp: `switch`

Die Roboterbewegung stoppt so schnell wie möglich und hält den TCP nahe an oder auf der Bahn (Softstopp), wenn der Wert des Suchsignals zu „aktiv“ oder der Wert der persistenten Variable zu TRUE wechselt. Der Roboter legt vor dem Stopp nur eine kurze Strecke zurück und wird nicht an die gesuchte Position zurück bewegt, d. h. an die Position, an der der Signalwechsel stattfand. `SStop` ist schneller als `PStop`. Wenn jedoch der Roboter die Geschwindigkeit von 100 mm/s

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

überschreitet, stoppt er tangential zur Bewegungsrichtung, was zu einer geringen Abweichung von der Bahn führt.

`[ \Sup ]`

Supervision

Datentyp: `switch`

Die Suchinstruktion reagiert empfindlich auf die Signalaktivierung oder eine Wertänderung der persistenten Variablen während der vollständigen Bewegung (fliegende Suche), d. h. sogar nachdem der erste Signalwechsel oder die erste Änderung einer persistenten Variablen gemeldet wurde. Wenn während einer Suche mehrere Übereinstimmungen gefunden werden, wird mit dem Roboter in `ToPoint` ein behebbarer Fehler generiert.

Wenn die Argumente `\Stop`, `\PStop`, `\SStop` und `\Sup` ausgelassen werden (es wird kein Schalter verwendet):

- Die Bewegung (fliegende Suche) an die im `ToPoint`-Argument angegebene Position wird fortgesetzt (wie bei Argument `\Sup`)
- Für keinen Suchtreffer wird ein Fehler gemeldet, doch für mehrere Suchtreffer wird kein Fehler gemeldet (der erste Suchtreffer wird als `SearchPoint` zurückgegeben.)

`Signal`

Datentyp: `signaldi`

Der Name des zu überwachenden Signals.

`PersBool`

Datentyp: `bool`

Die zu überwachende persistente Variable.

`[ \Flanks ]`

Datentyp: `switch`

Die positive und die negative Flanke des Signals gilt für ein Suchergebnis. Bei der Verwendung von Argument `PersBool` handelt es sich um die Wertänderung der Variable, die für ein Suchergebnis gültig ist.

Für Signal: Wenn das Argument `\Flanks` ausgelassen wird, gilt nur die positive Flanke des Signals für ein Suchergebnis, und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs einen positiven Wert hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (`ERR_SIGSUPSEARCH`) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Wenn das Argument `\Flanks` ausgelassen wird, gibt es nur bei einer Wertänderung zu TRUE ein gültiges Suchergebnis und eine Variablenüberwachung wird am Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs einen positiven Wert hat, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

`[ \PosFlank ]`

Datentyp: `switch`

Die positive Flanke des Signals oder die Wertänderung zu TRUE gilt für ein Suchergebnis, wenn eine persistente Variable verwendet wird.

`[ \NegFlank ]`

Datentyp: `switch`

Die negative Flanke des Signals oder die Wertänderung zu FALSE gilt für ein Suchergebnis, wenn eine persistente Variable verwendet wird.

`[ \HighLevel ]`

Datentyp: `switch`

Dieselbe Funktionalität, als ob der `\Flanks` Schalter nicht verwendet würde.

Für Signal: Die positive Flanke des Signals gilt für ein Suchergebnis und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs einen positiven Wert hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Nur die Wertänderung zu TRUE ist ein gültiges Suchergebnis und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs einen positiven Wert hat, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

`[ \LowLevel ]`

Datentyp: `switch`

Für Signal: Die negative Flanke des Signals gilt für ein Suchergebnis und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs den Wert 0 hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Nur die Wertänderung zu FALSE ist ein gültiges Suchergebnis und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs den Wert FALSE hat, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`SearchPoint`

Datentyp: `robtarget`

Die Position des TCP und der externen Achsen, wenn das Suchsignal ausgelöst wurde. Die Position wird im äußersten Koordinatensystem angegeben unter Berücksichtigung des angegebenen Werkzeugs, Werkobjekts und des aktiven `ProgDisp/ExtOffs`-Koordinatensystems.into consideration.

`CirPoint`

Datentyp: `robtarget`

Der Bogenpunkt des Roboters: Eine ausführlichere Beschreibung von kreisförmiger Bewegung finden Sie in der Erläuterung der Instruktion `MoveC`. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). `SearchC` verwendet stets einen Stopppunkt als Zonendaten für das Ziel.

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der externen Achsen und der Werkzeugumorientierung.

`[ \V ]`

Velocity

Datentyp: `num`

Dieses Argument wird verwendet, um die Geschwindigkeit des TCP in mm/s direkt in der Instruktion zu definieren. Es ersetzt dann die entsprechende Geschwindigkeit, die in den Geschwindigkeitsdaten angegeben ist.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.223 SearchC - Kreisförmige Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterpositionen in der Instruktion beziehen.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \Corr ]`

Correction

Datentyp: `switch`

Wenn dieses Argument vorhanden ist, werden die Korrekturdaten, die durch die Instruktion `CorrWrite` in einen Korrektureintrag geschrieben wurden, der Bahn und der Zielposition hinzugefügt.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

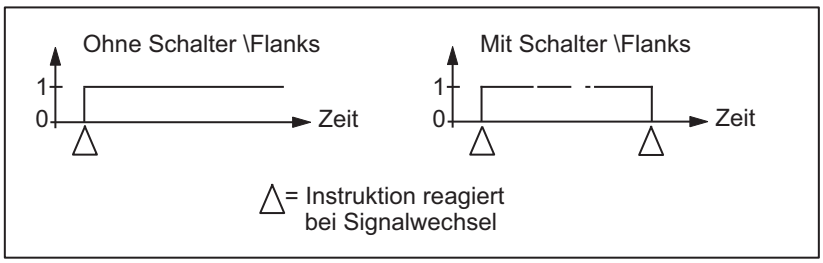
Siehe die Instruktion `MoveC` für weitere Informationen über kreisförmige Bewegung.

Die Bewegung endet stets mit einem Stopppunkt, d. h., der Roboter stoppt am Zielpunkt.

Bei einer fliegenden Suche, d. h., das Argument `\Sup` ist angegeben oder es ist kein Schalter angegeben, fährt der Roboter stets bis zum programmierten Zielpunkt fort. Wenn eine Suche mit dem Schalter `\Stop`, `\PStop` oder `\SStop` ausgeführt wird, stoppt die Roboterbewegung beim ersten Suchergebnis.

Die Instruktion `SearchC` gibt die Position des TCP zurück, wenn sich der Wert des digitalen Signals oder der persistenten Variablen in den angeforderten Wert ändert, wie in der folgenden Abbildung gezeigt.

Die Abbildung zeigt, wie die durch Flanken ausgelöste Signalerkennung verwendet wird (die Position wird nur beim ersten Signalwechsel gespeichert).



xx0500002237

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.223 SearchC - Kreisförmige Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_WHLSEARCH	• Keine Signalerkennung erfolgt. • Mehrere Signalerkennungen sind erfolgt. Dies tritt nur dann auf, wenn das Argument \Sup verwendet wird.
ERR_SIGSUPSEARCH	Das Signal hat bereits am Beginn des Suchvorgangs einen positiven Wert oder die Verbindung mit dem Signal ist nicht mehr vorhanden. Dies tritt nur dann auf, wenn das Argument \Flanks ausgelassen wird.
ERR_PERSSUPSEARCH	Die persistente Variable ist bereits zu Beginn des Suchprozesses TRUE. Dies tritt nur dann auf, wenn das Argument \Flanks ausgelassen wird.

Fehler können abhängig vom gewählten Abarbeitungsmodus auf verschiedene Weise behandelt werden:

- **Kontinuierlich vorwärts**/**Instruktion vorwärts**/ERR_WHLSEARCH: Es wird keine Position zurückgegeben und die Bewegung fährt stets zum programmierten Zielpunkt fort. Die Systemvariable ERRNO wird auf ERR_WHLSEARCH gesetzt und der Fehler kann in der Fehlerbehandlungsroutine der Routine behandelt werden.
- **Kontinuierlich vorwärts**/**Instruktion vorwärts**/ERR_SIGSUPSEARCH und ERR_PERSSUPSEARCH: Es wird keine Position zurückgegeben und die Bewegung stoppt so schnell wie möglich am Beginn der Suchstrecke. Die Systemvariable ERRNO wird abhängig vom verwendeten Argument (Signal oder persistente Variable) auf ERR_SIGSUPSEARCH oder ERR_PERSSUPSEARCH gesetzt und der Fehler kann in der Fehlerbehandlung dieser Routine behandelt werden.
- **Instruktion rückwärts**: Bei der Rückwärtsabarbeitung führt die Instruktion die Bewegung ohne jegliche Überwachung aus.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion SearchC sind unten aufgeführt.

Beispiel 1:

```
SearchC \Sup, di1\Flanks, sp, cirpoint, p10, v100, probe;
```

Der TCP der probe wird kreisförmig an die Position p10 bewegt. Wenn der Wert von Signal di1 zu aktiv oder passiv wechselt, wird die Position in sp gespeichert. Wenn sich der Signalwert zweimal ändert, generiert das Programm einen Fehler.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

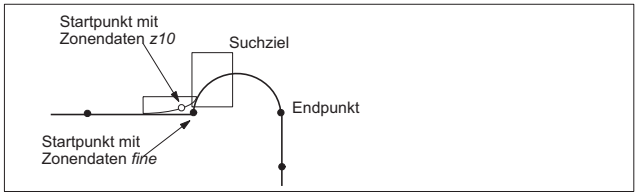
Allgemeine Einschränkungen gemäß der Instruktion `MoveC`.

`SearchC` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Zonendaten für die Positionierinstruktion, die `SearchC` vorangeht, müssen mit Umsicht verwendet werden. Empfohlen wird die Verwendung von `z0` oder einer kleinen Zone, die zu einer glatten Bewegung führt. Der Beginn der Suche, d. h. wenn das E/A-Signal reaktionsbereit ist, ist in diesem Fall nicht der programmierte Zielpunkt der vorherigen Positionierinstruktion, sondern ein Punkt an der tatsächlichen Roboterbahn. Die Abbildung unten zeigt ein Beispiel für einen möglichen Fehler, wenn andere Zonendaten als `fine`-Zonendaten verwendet werden.

Die Instruktion `SearchC` darf nie neu gestartet werden, nachdem der Kreispunkt passiert wurde. Anderenfalls arbeitet der Roboter die programmierte Bahn nicht weiter ab (die Positionierung an der kreisförmigen Bahn erfolgt nicht in der programmierten Richtung).

Die Abbildung zeigt, wie eine gefundene Position auf der falschen Seite des Objekts liegt, da die falschen Zonendaten verwendet wurden.



xx0500002238

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.223 SearchC - Kreisförmige Suchbewegung des Roboters

RobotWare - OS

Fortsetzung

WARNUNG

Einschränkungen für die Suche bei koordinierten Synchronbewegungen:

- Bei Verwendung von `SearchL`, `SearchC`oder `SearchExtJ` für eine Programmtask und einer anderen Bewegungsinstruktion für eine andere Programmtask kann die fliegende Suche nur mit dem Schalter `\Sup` ausgeführt werden. Außerdem kann die Fehlerbehandlung nur mit `TRYNEXT` ausgeführt werden.
- Es können alle Suchfunktionen verwendet werden, wenn einige der Instruktionen `SearchL`, `SearchC` oder `SearchExtJ` in allen beteiligten Programmtasks mit koordinierten Synchronbewegungen verwendet und Suchergebnisse aus demselben digitalen Eingangssignal generiert werden. Hierdurch werden in allen Suchinstruktionen Suchergebnisse synchron generiert. Außerdem muss die Fehlerbehandlung in allen Programmtasks identisch sein.

Während der Ausführung einer Suche kann die aktuelle Bahn nicht mit der Instruktion `StorePath` gespeichert werden.

Wiederholgenauigkeit für eine gefundene Position mit TCP-Geschwindigkeit 20-1000 mm/s beträgt 0,1-0,3 mm.

Typische Verzögerungsstrecke bei einer Suchgeschwindigkeit von 50 mm/s:

- ohne TCP auf der Bahn (Schalter `\Stop`) 1-3 mm
- mit TCP auf der Bahn (Schalter `\PStop`) 15-25 mm
- mit TCP auf der Bahn (Schalter `\SStop`) 4-8 mm

Einschränkungen beim Suchen eines Förderers:

- Bei einer Suche wird der Roboter angehalten, wenn das Objekt gefunden wird oder die Suche fehlschlägt. Führen Sie daher die Suche in der Bewegungsrichtung des Förderers aus, und fahren Sie nach dem Anhalten der Suche mit einer Bewegung an eine sichere Position fort. Verwenden Sie die Fehlerbehandlung, um den Roboter beim Fehlschlagen der Suche an eine sichere Position zu bewegen.
- Die Wiederholgenauigkeit für die Trefferposition der Suche ist bei der Suche für einen Förderer geringer und hängt von der Geschwindigkeit des Förderers sowie der Gleichmäßigkeit der Geschwindigkeit ab.

Syntax

```
SearchC
  ['\' Stop ','] | ['\' PStop ','] | ['\' SStop ','] | ['\' Sup
       ','']
  [Signal':=' ] <variable (VAR) of signaldi>|
  [PersBool ':=' ] <persistent (PERS) of bool>
  ['\' Flanks] |
  ['\' PosFlank] |
  ['\' NegFlank] |
  ['\' HighLevel] |
  ['\' LowLevel] ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[SearchPoint ':='] <var or pers (INOUT) of robtarget>','
[CirPoint ':='] <expression (IN) of robtarget>','
[ToPoint ':='] <expression (IN) of robtarget>','
['\' ID ':=' <expression (IN) of identno>]','
[Speed':='] <expression (IN) of speeddata>
['\' V ':=' <expression (IN) of num>]|
['\' T ':=' <expression (IN) of num>]','
[Tool ':='] <persistent (PERS) of tooldata>
['\' WObj':=' <persistent (PERS) of wobjdata>]
['\' Corr ]
['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Lineare Suchvorgänge	*SearchL - Lineare Suchbewegung des Roboters auf Seite 681*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Kreisförmige Bewegung des Roboters	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Kreisbewegung	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Verwenden von Fehlerbehandlungsroutinen	*Technisches Referenzhandbuch - RAPID Overview*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.224 SearchExtJ - Suchen mit einer oder mehreren mechanischer Einheiten ohne TCP

Verwendung

`SearchExtJ` (*Search External Joints*) wird verwendet, um die Position einer externen Achse zu suchen, wenn nur externe lineare oder externe Rotationsachsen bewegt werden. Die externen Achsen können zu einer oder mehreren mechanischen Einheiten ohne TCP gehören.

Während der Bewegung überwacht der Roboter ein digitales Eingangssignal oder eine persistente Variable. Wenn das Signal oder die persistente Variable den erforderlichen Wert annimmt, liest der Roboter sofort die aktuelle Position ab.

Diese Instruktion kann nur unter folgenden Bedingungen verwendet werden:

- Die aktuelle Programmtask ist als Bewegungstask definiert
- Die Task steuert eine oder mehrere mechanische Einheiten ohne TCP

Bei Verwendung von Suchinstruktionen ist es wichtig, das E/A-System so zu konfigurieren, dass zwischen dem Setzen des physischen Signals und dem Ankommen der Informationen über das Setzen des Signals im System nur eine sehr kurze Zeitverzögerung liegt. (Verwenden Sie das E/A-Gerät mit Interruptsteuerung, nicht mit Abfragesteuerung.) Die Vorgehensweise kann von Feldbus zu Feldbus unterschiedlich sein. Bei der Verwendung von DeviceNet sind in den ABB-Einheiten DSQC 651 (AD Kombi E/A) und DSQC 652 (Digitale E/A) kurze zeitliche Verzögerung vorgesehen, da sie den Verbindungstyp „Change of State“ (Statuswechsel) verwenden. Bei Verwendung anderer Feldbusse muss das Netzwerk ordnungsgemäß konfiguriert werden, sodass die richtigen Bedingungen vorliegen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SearchExtJ`:

Siehe auch *Weitere Beispiele auf Seite 678*.

Beispiel 1:

```
SearchExtJ di1, searchp, jpos10, vrot20;
```

Die mechanische Einheit mit Rotationsachsen wird mit der Geschwindigkeit `vrot20` zur Position `jpos10` bewegt. Wenn der Wert von Signal `di1` zu aktiv wechselt, wird die Position in `searchp` gespeichert.

Beispiel 2:

```
SearchExJ \Stop, di2, posx, jpos20, vlin50;
```

Die mechanische Einheit mit linearer Achse wird zur Position `jpos20` bewegt. Wenn der Wert von Signal `di2` zu aktiv wechselt, wird die Position in `posx` gespeichert und die Bewegung wird sofort angehalten.

Beispiel 3

```
PERS bool mypers:=FALSE;
...
SearchExJ \Stop, di2, posx, jpos20, vlin50;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.224 SearchExtJ - Suchen mit einer oder mehreren mechanischer Einheiten ohne TCP

Die mechanische Einheit mit linearer Achse wird zur Position `jpos20` bewegt. Wenn der Wert der persistenten Variable `mypers` zu TRUE wechselt, wird die Position in `posx` gespeichert und die Bewegung wird sofort angehalten.

Argumente

```
SearchExtJ [\Stop] | [\PStop] | [\SStop] | [\Sup] Signal | PersBool
     [\Flanks] | [\PosFlank] | [\NegFlank] | [\HighLevel] |
     [\LowLevel] SearchJointPos ToJointPos [\ID] [\UseEOffs] Speed
     [\T]
```

`[ \Stop ]`

Stiff Stop

Datentyp: `switch`

Die Bewegung stoppt so schnell wie möglich mit einem harten Stopp, wenn der Wert des Suchsignals zu „aktiv“ wechselt oder der Wert der persistenten Variable zu TRUE wechselt. Die externen Achsen legen vor dem Stopp eine kurze Strecke zurück und werden nicht an die gesuchte Position zurückbewegt, d. h. an die Position, an der der Signalwechsel stattfand.

`[ \PStop ]`

Path Stop

Datentyp: `switch`

Die Bewegung stoppt so schnell wie möglich mit Path Stop (Programmstopp), wenn der Wert des Suchsignals zu „aktiv“ wechselt oder der Wert der persistenten Variable zu TRUE wechselt. Die externen Achsen legen vor dem Stopp eine relativ lange Strecke zurück und werden nicht an die gesuchte Position zurückbewegt, d. h. an die Position, an der der Signalwechsel stattfand.

`[ \SStop ]`

Soft Stop

Datentyp: `switch`

Die Bewegung stoppt so schnell wie möglich mit einem schnellen Softstopp, wenn der Wert des Suchsignals zu „aktiv“ wechselt oder der Wert der persistenten Variable zu TRUE wechselt. Die externen Achsen legen vor dem Stopp nur eine kurze Strecke zurück und werden nicht an die gesuchte Position zurückbewegt, d. h. an die Position, an der der Signalwechsel stattfand.

`Stop` ist schneller als `SStop`. `SStop` ist schneller als `PStop`.

`[ \Sup ]`

Supervision

Datentyp: `switch`

Die Suchinstruktion reagiert empfindlich auf die Signalaktivierung oder eine Wertänderung der persistenten Variablen während der vollständigen Bewegung (fliegende Suche), d. h. sogar nachdem der erste Signalwechsel oder die erste Änderung einer persistenten Variablen gemeldet wurde. Wenn während einer Suche mehrere Übereinstimmungen gefunden werden, wird mit dem Roboter in `ToPoint` ein behebbarer Fehler generiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.224 SearchExtJ - Suchen mit einer oder mehreren mechanischer Einheiten ohne TCP
RobotWare - OS
Fortsetzung

Wenn die Argumente \Stop, \PStop, \SStop und \Sup ausgelassen werden (es wird kein Schalter verwendet):

- Die Bewegung (fliegende Suche) an die im ToJointPos-Argument angegebene Position wird fortgesetzt (wie bei Argument \Sup).
- Für einen Suchtreffer wird ein Fehler gemeldet, doch für mehrere Suchtreffer wird kein Fehler gemeldet (der erste Suchtreffer wird als SearchJointPos zurückgegeben).

Signal

Datentyp: signaldi

Der Name des zu überwachenden Signals.

PersBool

Datentyp: bool

Die zu überwachende persistente Variable.

[\Flanks]

Datentyp: switch

Die positive und die negative Flanke des Signals gilt für ein Suchergebnis. Bei der Verwendung von Argument PersBool handelt es sich um die Wertänderung der Variable, die für ein Suchergebnis gültig ist.

Für Signal: Wenn das Argument \Flanks ausgelassen wird, gilt nur die positive Flanke des Signals für ein Suchergebnis, und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs einen positiven Wert hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Bewegung so schnell wie möglich gestoppt. Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Wenn das Argument \Flanks ausgelassen wird, gibt es nur bei einer Wertänderung zu TRUE ein gültiges Suchergebnis, und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs einen positiven Wert hat, wird die Bewegung so schnell wie möglich gestoppt. Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

[\PosFlank]

Datentyp: switch

Die positive Flanke des Signals oder die Wertänderung zu TRUE gilt für ein Suchergebnis, wenn eine persistente Variable verwendet wird.

[\NegFlank]

Datentyp: switch

Die negative Flanke des Signals oder die Wertänderung zu FALSE gilt für ein Suchergebnis, wenn eine persistente Variable verwendet wird.

[\HighLevel]

Datentyp: switch

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dieselbe Funktionalität, als ob der \Flanks Schalter nicht verwendet würde.

Für Signal: Die positive Flanke des Signals gilt für ein Suchergebnis und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs einen positiven Wert hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Bewegung so schnell wie möglich gestoppt. Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Nur die Wertänderung zu TRUE ist ein gültiges Suchergebnis und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs einen positiven Wert hat, wird die Bewegung so schnell wie möglich gestoppt. Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

[\LowLevel]

Datentyp: switch

Für Signal: Die negative Flanke des Signals gilt für ein Suchergebnis und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs den Wert 0 hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Bewegung so schnell wie möglich gestoppt. Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Nur die Wertänderung zu FALSE ist ein gültiges Suchergebnis und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs den Wert TRUE hat, wird die Bewegung so schnell wie möglich gestoppt. Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

SearchJointPos

Datentyp: jointtarget

Die Position der externen Achsen zu dem Zeitpunkt, zu dem das Suchsignal ausgelöst wurde. Bei der Position wird ein aktives ExtOffs ggf. berücksichtigt.

ToJointPos

Datentyp: jointtarget

Der Zielpunkt der externen Achsen. Sie wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). SearchExtJ verwendet immer einen Stopppunkt als Zonendaten für das Ziel.

[\ID]

Synchronization id

Datentyp: identno

Das Argument [\ID] ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`[ \UseEOffs ]`

Use External Offset

Datentyp: `switch`

Der mit der Instruktion `EOffsSet` eingerichtete Offset für externe Achsen wird für die `SearchExtJ`-Instruktion aktiviert, wenn das Argument `UseEOffs` verwendet wird. Weitere Informationen über externe Offsets finden Sie in den Erläuterungen zur Instruktion `EOffsSet`.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Diese Daten definieren die Geschwindigkeit für die externen linearen und die externen Rotationsachsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument gibt die Gesamtdauer in Sekunden an, für die sich die mechanischen Einheiten bewegen. Es ersetzt dann die entsprechenden Geschwindigkeitsdaten.

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der die mechanische Einheit sich bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn die mechanische Einheit diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Programmabarbeitung

Siehe die Instruktion `MoveExtJ` für Informationen über Bewegungen mechanischer Einheiten ohne TCP.

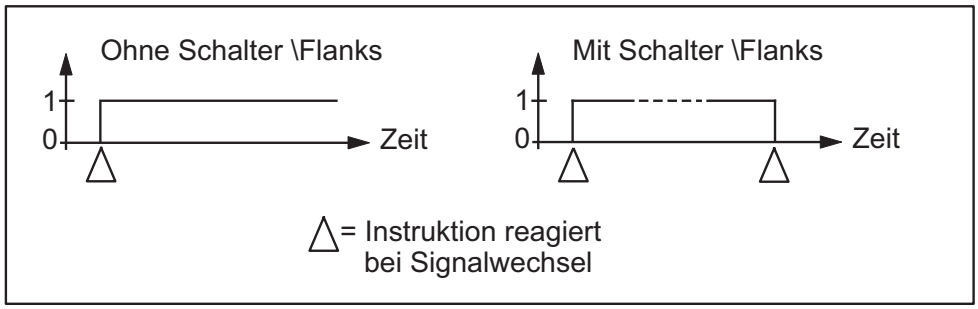
Die Bewegung endet stets mit einem Stopppunkt, d. h., die externen Achsen stoppen am Zielpunkt. Bei einer fliegenden Suche, d. h., das Argument `\Sup` ist angegeben oder es ist kein Schalter angegeben, wird die Bewegung stets bis zum programmierten Zielpunkt fortgesetzt. Wenn eine Suche mit dem Schalter `\Stop`, `\PStop` oder `\SStop` ausgeführt wird, stoppt die Bewegung beim ersten Suchergebnis.

Die Instruktion `SearchExtJ` speichert die Position der externen Achsen, wenn sich der Wert des digitalen Signals oder der persistenten Variable in den angeforderten Wert ändert, wie in der folgenden Abbildung gezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Abbildung zeigt, wie die durch Flanken ausgelöste Signalerkennung verwendet wird (die Position wird nur beim ersten Signalwechsel gespeichert).



xx0500002243

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_WHLSEARCH	• Keine Signalerkennung erfolgt. • Mehrere Signalerkennungen sind erfolgt. Dies tritt nur dann auf, wenn das Argument \Sup verwendet wird.
ERR_SIGSUPSEARCH	Das Signal hat bereits am Beginn des Suchvorgangs einen positiven Wert oder die Verbindung mit dem Signal ist nicht mehr vorhanden. Dies tritt nur dann auf, wenn das Argument \Flanks ausgelassen wird.
ERR_PERSSUPSEARCH	Die persistente Variable ist bereits zu Beginn des Suchprozesses TRUE. Dies tritt nur dann auf, wenn das Argument \Flanks ausgelassen wird.

Fehler können abhängig vom gewählten Abarbeitungsmodus auf verschiedene Weise behandelt werden:

- **Kontinuierlich vorwärts**/**Instruktion vorwärts**/ERR_WHLSEARCH: Es wird keine Position zurückgegeben und die Bewegung fährt stets zum programmierten Zielpunkt fort. Die Systemvariable ERRNO wird auf ERR_WHLSEARCH gesetzt und der Fehler kann in der Fehlerbehandlungsroutine der Routine behandelt werden.
- **Kontinuierlich vorwärts**/**Instruktion vorwärts**/ERR_SIGSUPSEARCH und ERR_PERSSUPSEARCH: Es wird keine Position zurückgegeben und die Bewegung stoppt so schnell wie möglich am Beginn der Suchstrecke. Die Systemvariable ERRNO wird abhängig vom verwendeten Argument (Signal oder persistente Variable) auf ERR_SIGSUPSEARCH oder ERR_PERSSUPSEARCH gesetzt und der Fehler kann in der Fehlerbehandlung dieser Routine behandelt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.224 SearchExtJ - Suchen mit einer oder mehreren mechanischer Einheiten ohne TCP
RobotWare - OS
Fortsetzung

- **Instruktion rückwärts**: Bei der Rückwärtsabarbeitung führt die Instruktion die Bewegung ohne jegliche Überwachung aus.

Beispiel

```
VAR num fk;
...
MoveExtJ jpos10, vrot100, fine;
SearchExtJ \Stop, di1, sp, jpos20, vrot5;
...
ERROR
  IF ERRNO=ERR_WHLSEARCH THEN
    StorePath;
    MoveExtJ jpos10, vrot50, fine;
    RestoPath;
    ClearPath;
    StartMove;
    RETRY;
  ELSEIF ERRNO=ERR_SIGSUPSEARCH THEN
    TPWrite "The signal of the SearchExtJ instruction is already
         high!";
    TPReadFK fk,"Try again after manual reset of signal ?","YES",
         stEmpty, stEmpty, stEmpty, "NO";
    IF fk = 1 THEN
      StorePath;
      MoveExtJ jpos10, vrot50, fine;
      RestoPath;
      ClearPath;
      StartMove;
      RETRY;
    ELSE
      Stop;
    ENDIF
  ENDIF
```

Wenn das Signal bereits am Beginn des Suchvorgangs aktiv ist, wird ein Anwenderdialog aktiviert (`TPReadFK ...;`). Setzen Sie das Signal zurück und klicken Sie im Anwenderdialog auf JA, damit die mechanische Einheit zurück zu `jpos10` fährt und es erneut versucht. Andernfalls stoppt die Programmabarbeitung.

Wenn das Signal zu Beginn des Suchvorgangs passiv ist, sucht die mechanische Einheit von Position `jpos10` bis `jpos20`. Wenn keine Signalerkennung erfolgt, bewegt sich der Roboter zurück zu `jpos10` und versucht es erneut.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `SearchExtJ` sind unten aufgeführt.

Beispiel 1:

```
SearchExtJ \Sup, di1\Flanks, searchp,jpos10, vrot20;
```

Die mechanische Einheit wird zur Position `jpos10` bewegt. Wenn der Wert von Signal `di1` zu aktiv oder passiv wechselt, wird die Position in `searchp`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

gespeichert. Wenn sich der Signalwert zweimal ändert, generiert das Programm nach Abschluss des Suchvorgangs einen Fehler.

Beispiel 2:

```
SearchExtJ \Stop, di1, sp, jpos20, vlin50;
MoveExtJ sp, vlin50, fine \Inpos := inpos50;
```

Zu Beginn des Suchvorgangs erfolgt eine Prüfung des Signals `dil`, und wenn das Signal bereits einen positiven Wert hat oder wenn die Verbindung mit dem Signal getrennt wurde, stoppt die Bewegung. Anderenfalls wird die mechanische Einheit zur Position `jpos20` bewegt. Wenn sich der Wert des Signals `di1` in aktiv ändert, wird die Position in `sp` gespeichert. Die mechanische Einheit wird mit einem genau definierten Stopppunkt zurück zu diesem Punkt bewegt.

Einschränkungen

`SearchExtJ` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Einschränkungen für die Suche bei koordinierten Synchronbewegungen:

- Bei Verwendung von `SearchL`, `SearchC` oder `SearchExtJ` für eine Programmtask und einer anderen Bewegungsinstruktion für eine andere Programmtask kann die fliegende Suche nur mit dem Schalter `\Sup` ausgeführt werden. Außerdem kann die Fehlerbehandlung nur mit `TRYNEXT` ausgeführt werden.
- Es können alle Suchfunktionen verwendet werden, wenn einige der Instruktionen `SearchL`, `SearchC` oder `SearchExtJ` in allen beteiligten Programmtasks mit koordinierten Synchronbewegungen verwendet und Suchergebnisse aus demselben digitalen Eingangssignal generiert werden. Hierdurch werden in allen Suchinstruktionen Suchergebnisse synchron generiert. Außerdem muss die Fehlerbehandlung in allen Programmtasks identisch sein.
- Während der Ausführung einer Suche kann die aktuelle Bahn nicht mit der Instruktion `StorePath` gespeichert werden.

Syntax

```
SearchExtJ
  ['\' Stop ','] | ['\' PStop ','] | ['\' SStop ','] | ['\' Sup
       ',']
  [Signal ':='] <variable (VAR) of signaldi> |
  [PersBool ':='] <persistent (PERS) of bool>
  ['\' Flanks] |
  ['\' PosFlank] |
  ['\' NegFlank] |
  ['\' HighLevel] |
  ['\' LowLevel] ','
  [SearchJointPos ':='] <var or pers (INOUT) of jointtarget>','
  [ToJointPos ':=' ] <expression (IN) of jointtarget >
  ['\' ID ':=' <expression (IN) of identno >]','
  ['\' UseEOffs ',' ]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.224 SearchExtJ - Suchen mit einer oder mehreren mechanischer Einheiten ohne TCP

RobotWare - OS

Fortsetzung

```
[Speed ':=' ] <expression (IN) of speeddata>
['\' T ':=' <expression (IN) of num> ]';'
```

Weitere Informationen

Information über	Siehe
Bewegen von mechanischen Einheiten ohne TCP	*MoveExtJ - Bewegen einer oder mehrerer mechanischer Einheiten ohne TCP auf Seite 418*
Definition von jointtarget	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Verwenden von Fehlerbehandlungsroutinen	*Technisches Referenzhandbuch - RAPID Overview*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters

Verwendung

`SearchL` (*Search Linear*) wird verwendet, um mit einer linearen Bewegung des Werkzeugarbeitspunkts (TCP) eine Position zu suchen.

Während der Bewegung überwacht der Roboter ein digitales Eingangssignal oder eine persistente Variable. Wenn das Signal oder die persistente Variable den erforderlichen Wert annimmt, liest der Roboter sofort die aktuelle Position ab.

Diese Instruktion eignet sich typischerweise für den Einsatz eines Sensors zur Oberflächenerkennung. Mit der Instruktion `SearchL` kann der Umriss eines Werkobjekts ermittelt werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Bei Verwendung von Suchinstruktionen ist es wichtig, das E/A-System so zu konfigurieren, dass zwischen dem Setzen des physischen Signals und dem Ankommen der Informationen über das Setzen im System nur eine sehr kurze Zeitspanne liegt. (Verwenden Sie die E/A-Gerät mit Interruptsteuerung, nicht mit Abfragesteuerung.) Die Vorgehensweise kann von Feldbus zu Feldbus unterschiedlich sein. Bei der Verwendung von DeviceNet sind in den ABB-Einheiten (lokale E/A) kurze Intervalle vorgesehen, da sie den Verbindungstyp *Statuswechsel* verwenden. Achten Sie bei der Verwendung anderer Feldbusse darauf, dass das Netzwerk ordnungsgemäß konfiguriert ist, sodass die richtigen Bedingungen vorliegen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SearchL`:

Siehe auch *Weitere Beispiele auf Seite 689*.

Beispiel 1:

```
SearchL di1, sp, p10, v100, probe;
```

Der TCP der `probe` wird mit der Geschwindigkeit `v100` linear an die Position `p10` bewegt. Wenn der Wert von Signal `di1` zu aktiv wechselt, wird die Position in `sp` gespeichert.

Beispiel 2:

```
SearchL \Stop, di2, sp, p10, v100, probe;
```

Der TCP der `probe` wird linear an die Position `p10` bewegt. Wenn der Wert von Signal `di2` zu aktiv wechselt, wird die Position in `sp` gespeichert und der Roboter stoppt sofort.

Beispiel 3

```
PERS bool mypers:=FALSE;
...
SearchL mypers, sp, p10, v100, probe;
```

Der TCP der `probe` wird mit der Geschwindigkeit `p10` linear an die Position `v100` bewegt. Wenn der Wert der persistenten Variable `mypers` zu TRUE wechselt, wird die Position in `sp` gespeichert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

Argumente

```
SearchL [\Stop] | [\PStop] | [\SStop] | [\Sup] Signal | PersBool
     [\Flanks] | [\PosFlank] | [\NegFlank] | [\HighLevel] |
     [\LowLevel] SearchPoint ToPoint [\ID] Speed [\V] | [\T] Tool
     [\WObj] [\Corr] [\TLoad]
```

`[ \Stop ]`

Stiff Stop

Datentyp: `switch`

Die Roboterbewegung stoppt so schnell wie möglich, ohne den TCP auf der Bahn zu halten (harter Stopp), wenn der Wert des Suchsignals zu „aktiv“ oder der Wert der persistenten Variable zu TRUE wechselt. Der Roboter legt jedoch vor dem Stopp eine kurze Strecke zurück und wird nicht an die gesuchte Position zurückbewegt, d. h. an die Position, wo sich das Signal oder der persistente Wert geändert hat.

Testen Sie zunächst eine langsame Geschwindigkeit empfohlen, <100 Beispiel mm/s, die dann schrittweise auf den gewünschten Wert erhöht wird.

WARNUNG

Ein Suchstopp durch einen abrupten Stopp (Schalter `\Stop`) darf nur bei einer TCP-Geschwindigkeit von weniger als 100 mm/s durchgeführt werden. Bei einem abrupten Stopp bei höherer Geschwindigkeit können sich die Achsen möglicherweise unkontrolliert bewegen.

Hinweis

Für einen YuMi-Roboter beträgt die Höchstgeschwindigkeit für Suchen mit abrupten Stopp 1000 mm/s.

`[ \PStop ]`

Path Stop

Datentyp: `switch`

Die Roboterbewegung stoppt so schnell wie möglich, wobei der TCP auf der Bahn gehalten wird (Softstopp), wenn der Wert des Suchsignals zu „aktiv“ oder der Wert der persistenten Variable zu TRUE wechselt. Der Roboter legt jedoch vor dem Stopp eine Strecke zurück und wird nicht an die gesuchte Position zurückbewegt, d. h. an die Position, wo sich das Signal oder der persistente Wert geändert hat.

`[ \SStop ]`

Soft Stop

Datentyp: `switch`

Die Roboterbewegung stoppt so schnell wie möglich und hält den TCP nahe an oder auf der Bahn (Softstopp), wenn der Wert des Suchsignals zu „aktiv“ wechselt oder der Wert der persistenten Variable zu TRUE wechselt. Der Roboter legt vor dem Stopp nur eine kurze Strecke zurück und wird nicht an die gesuchte Position zurückbewegt, d. h. an die Position, an der der Signalwechsel oder der Wechsel der persistenten Variablen stattfand. `SStop` ist schneller als `PStop`. Wenn jedoch

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

der Roboter die Geschwindigkeit von 100 mm/s überschreitet, stoppt er tangential zur Bewegungsrichtung, was zu einer geringen Abweichung von der Bahn führt.

`[ \Sup ]`

Supervision

Datentyp: `switch`

Die Suchinstruktion reagiert empfindlich auf die Signalaktivierung oder eine Wertänderung der persistenten Variablen während der vollständigen Bewegung (fliegende Suche), d. h. sogar nachdem der erste Signalwechsel oder die erste Änderung einer persistenten Variablen gemeldet wurde. Wenn während einer Suche mehrere Übereinstimmungen gefunden werden, wird mit dem Roboter in `ToPoint` ein behebbarer Fehler generiert.

Wenn die Argumente `\Stop`, `\PStop`, `\SStop` und `\Sup` ausgelassen werden (es wird kein Schalter verwendet):

- Die Bewegung (fliegende Suche) an die im `ToPoint`-Argument angegebene Position wird fortgesetzt (wie bei Argument `\Sup`)
- Für keinen Suchtreffer wird ein Fehler gemeldet, doch für mehrere Suchtreffer wird kein Fehler gemeldet (der erste Suchtreffer wird als `SearchPoint` zurückgegeben.)

`Signal`

Datentyp: `signaldi`

Der Name des zu überwachenden Signals.

`PersBool`

Datentyp: `bool`

Die zu überwachende persistente Variable.

`[ \Flanks ]`

Datentyp: `switch`

Die positive und die negative Flanke des Signals gilt für ein Suchergebnis. Bei der Verwendung von Argument `PersBool` handelt es sich um die Wertänderung der Variable, die für ein Suchergebnis gültig ist.

Für Signal: Wenn das Argument `\Flanks` ausgelassen wird, gilt nur die positive Flanke des Signals für ein Suchergebnis, und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs einen positiven Wert hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (`ERR_SIGSUPSEARCH`) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Wenn das Argument `\Flanks` ausgelassen wird, gibt es nur bei einer Wertänderung zu TRUE ein gültiges Suchergebnis und eine Variablenüberwachung wird am Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs einen positiven Wert hat, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

[\PosFlank]

Datentyp: switch

Die positive Flanke des Signals oder die Wertänderung zu TRUE gilt für ein Suchergebnis, wenn eine persistente Variable verwendet wird.

[\NegFlank]

Datentyp: switch

Die negative Flanke des Signals oder die Wertänderung zu FALSE gilt für ein Suchergebnis, wenn eine persistente Variable verwendet wird.

[\HighLevel]

Datentyp: switch

Dieselbe Funktionalität, als ob der \Flanks Schalter nicht verwendet würde.

Für Signal: Die positive Flanke des Signals gilt für ein Suchergebnis und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs einen positiven Wert hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Nur die Wertänderung zu TRUE ist ein gültiges Suchergebnis und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs einen positiven Wert hat, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

[\LowLevel]

Datentyp: switch

Für Signal: Die negative Flanke des Signals gilt für ein Suchergebnis und eine Signalüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also das Signal bereits zu Beginn des Suchvorgangs den Wert 0 hat oder die Verbindung mit dem Signal unterbrochen wird, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_SIGSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Für persistente Variable: Nur die Wertänderung zu FALSE ist ein gültiges Suchergebnis und eine Variablenüberwachung wird zu Beginn des Suchvorgangs aktiviert. Wenn also die persistente Variable bereits zu Beginn des Suchvorgangs den Wert FALSE hat, wird die Roboterbewegung so schnell wie möglich gestoppt und der TCP auf der Bahn gehalten (Softstopp). Ein Wiederherstellungsfehler (ERR_PERSSUPSEARCH) wird erzeugt, der in der Fehlerbehandlungsroutine behoben werden kann.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`SearchPoint`

Datentyp: `robtarget`

Die Position des TCP und der externen Achsen, wenn das Suchsignal ausgelöst wurde. Die Position wird im äußersten Koordinatensystem angegeben unter Berücksichtigung des angegebenen Werkzeugs, Werkobjekts und des aktiven `ProgDisp/ExtOffs`-Koordinatensystems.

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet). `SearchL` verwendet stets einen Stopppunkt als Zonendaten für das Ziel.

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der externen Achsen und der Werkzeugumorientierung.

`[ \V ]`

Velocity

Datentyp: `num`

Dieses Argument wird verwendet, um die Geschwindigkeit des TCP in mm/s direkt in der Instruktion zu definieren. Es ersetzt dann die entsprechende Geschwindigkeit, die in den Geschwindigkeitsdaten angegeben ist.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \Corr ]`

Correction

Datentyp: `switch`

Korrekturdaten, die durch die Instruktion `CorrWrite` in einen Korrektureintrag geschrieben wurden, werden der Bahn und der Zielposition hinzugefügt, falls dieses Argument vorhanden ist.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

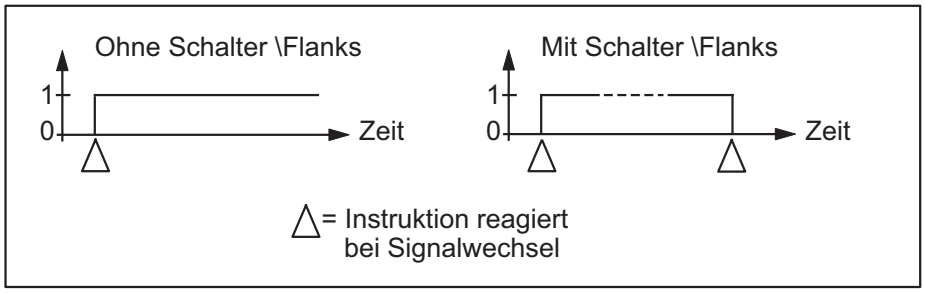
Programmabarbeitung

Siehe die Instruktion `MoveL` für weitere Informationen über lineare Bewegung.

Die Bewegung endet stets mit einem Stopppunkt, d. h., der Roboter stoppt am Zielpunkt. Bei einer fliegenden Suche, d. h., das Argument `\Sup` ist angegeben oder es ist kein Schalter angegeben, fährt der Roboter stets bis zum programmierten Zielpunkt fort. Wenn eine Suche mit dem Schalter `\Stop`, `\PStop` oder `\SStop` ausgeführt wird, stoppt die Roboterbewegung beim ersten Suchergebnis.

Die Instruktion `SearchL` speichert die Position des TCP, wenn sich der Wert des digitalen Signals oder der persistenten Variablen in den angeforderten Wert ändert, wie in der folgenden Abbildung gezeigt.

Die Abbildung zeigt, wie die durch Flanken ausgelöste Signalerkennung verwendet wird (die Position wird nur beim ersten Signalwechsel gespeichert).



xx0500002243

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_WHLSEARCH`	• Keine Signalerkennung erfolgt. • Mehrere Signalerkennungen sind erfolgt. Dies tritt nur dann auf, wenn das Argument `\Sup` verwendet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_SIGSUPSEARCH	Das Signal hat bereits am Beginn des Suchvorgangs einen positiven Wert oder die Verbindung mit dem Signal ist nicht mehr vorhanden. Dies tritt nur dann auf, wenn das Argument \Flanks ausgelassen wird.
ERR_PERSSUPSEARCH	Die persistente Variable ist bereits zu Beginn des Suchprozesses TRUE. Dies tritt nur dann auf, wenn das Argument \Flanks ausgelassen wird.

Fehler können abhängig vom gewählten Abarbeitungsmodus auf verschiedene Weise behandelt werden:

- **Kontinuierlich vorwärts/Instruktion vorwärts**/ERR_WHLSEARCH: Es wird keine Position zurückgegeben und die Bewegung fährt stets zum programmierten Zielpunkt fort. Die Systemvariable ERRNO wird auf ERR_WHLSEARCH gesetzt und der Fehler kann in der Fehlerbehandlungsroutine der Routine behandelt werden.
- **Kontinuierlich vorwärts/Instruktion vorwärts**/ERR_SIGSUPSEARCH und ERR_PERSSUPSEARCH: Es wird keine Position zurückgegeben und die Bewegung stoppt so schnell wie möglich am Beginn der Suchstrecke. Die Systemvariable ERRNO wird abhängig vom verwendeten Argument (Signal oder persistente Variable) auf ERR_SIGSUPSEARCH oder ERR_PERSSUPSEARCH gesetzt und der Fehler kann in der Fehlerbehandlung dieser Routine behandelt werden.
- **Instruktion rückwärts**: Bei der Rückwärtsabarbeitung führt die Instruktion die Bewegung ohne jegliche Überwachung aus.

Beispiel

```
VAR num fk;
...
MoveL p10, v100, fine, tool1;
SearchL \Stop, di1, sp, p20, v100, tool1;
...
ERROR
  IF ERRNO=ERR_WHLSEARCH THEN
    StorePath;
    MoveL p10, v100, fine, tool1;
    RestoPath;
    ClearPath;
    StartMove;
    RETRY;
  ELSEIF ERRNO=ERR_SIGSUPSEARCH THEN
    TPWrite "The signal of the SearchL instruction is already
         high!";
    TPReadFK fk,"Try again after manual reset of signal ?","YES",
         stEmpty, stEmpty, stEmpty, "NO";
    IF fk = 1 THEN
      StorePath;
      MoveL p10, v100, fine, tool1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
      RestoPath;
      ClearPath;
      StartMove;
      RETRY;
    ELSE
      Stop;
    ENDIF
  ENDIF
```

Wenn das Signal bereits am Beginn des Suchvorgangs aktiv ist oder wenn die Verbindung mit dem Signal unterbrochen ist, wird ein Anwenderdialog aktiviert (`TPReadFK ...;`). Setzen Sie das Signal zurück und klicken Sie im Anwenderdialog auf JA, damit der Roboter zurück zu `p10` fährt und es erneut versucht. Andernfalls stoppt die Programmabarbeitung.

Wenn das Signal zu Beginn des Suchvorgangs passiv ist, sucht der Roboter von Position `p10` bis `p20`. Wenn keine Signalerkennung erfolgt, bewegt sich der Roboter zurück zu `p10` und versucht es erneut.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `SearchL` sind unten aufgeführt.

Beispiel 1:

```
SearchL \Sup, di1 \Flanks, sp, p10, v100, probe;
```

Der TCP der `probe` wird linear an die Position `p10` bewegt. Wenn der Wert von Signal `di1` zu aktiv oder passiv wechselt, wird die Position in `sp` gespeichert. Wenn sich der Signalwert zweimal ändert, generiert das Programm nach Abschluss des Suchvorgangs einen Fehler.

Beispiel 2:

```
SearchL \Stop, di1, sp, p10, v100, tool1;
MoveL sp, v100, fine \Inpos := inpos50, tool1;
PDispOn *, tool1;
MoveL p100, v100, z10, tool1;
MoveL p110, v100, z10, tool1;
MoveL p120, v100, z10, tool1;
PDispOff;
```

Zu Beginn des Suchvorgangs erfolgt eine Prüfung des Signals `di1` und wenn das Signal bereits einen positiven Wert hat oder wenn die Verbindung mit dem Signal getrennt wurde, stoppt der Roboter. Andernfalls wird der TCP von `tool1` linear an die Position `p10` bewegt. Wenn sich der Wert des Signals `di1` in aktiv ändert, wird die Position in `sp.` gespeichert. Der Roboter wird mit einem genau definierten Stopppunkt zurück zu diesem Punkt bewegt. Mithilfe von Programmverschiebung bewegt sich der Roboter dann relativ zur gesuchten Position `sp`.

Beispiel 3

```
PERS bool MyTrigger:=FALSE;
...
SearchL \Stop, MyTrigger, sp, p10, v100, tool1;
MoveL sp, v100, fine \Inpos := inpos50, tool1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters

RobotWare - OS

Fortsetzung

```
PDispOn *, tool1;
MoveL p100, v100, z10, tool1;
MoveL p110, v100, z10, tool1;
MoveL p120, v100, z10, tool1;
PDispOff;
```

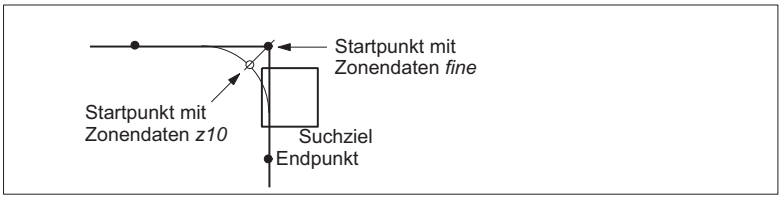
Zu Beginn des Suchvorgangs erfolgt eine Prüfung der persistenten Variable `MyTrigger` und wenn die Variable bereits den Wert TRUE hat, stoppt der Roboter. Andernfalls wird der TCP von `tool1` linear an die Position `p10` bewegt. Wenn sich der Wert der persistenten Variable `MyTrigger` in TRUE ändert, wird die Position in `sp.` gespeichert. Der Roboter wird mit einem genau definierten Stopppunkt zurück zu diesem Punkt bewegt. Mithilfe von Programmverschiebung bewegt sich der Roboter dann relativ zur gesuchten Position `sp`.

Einschränkungen

`SearchL` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

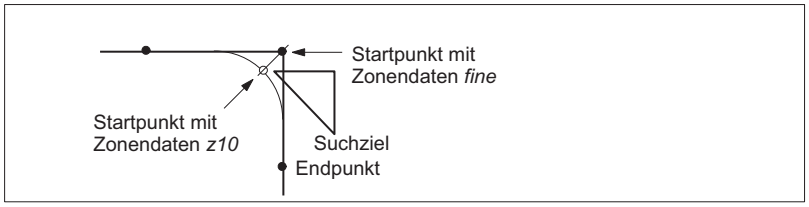
Zonendaten für die Positionierinstruktion, die `SearchL` vorangeht, müssen mit Umsicht verwendet werden. Der Beginn der Suche, d. h. wenn das E/A-Signal reaktionsbereit ist, ist in diesem Fall nicht der programmierte Zielpunkt der vorherigen Positionierinstruktion, sondern ein Punkt an der tatsächlichen Roboterbahn. Die Abbildungen unten zeigen Beispiele für mögliche Fehler, wenn andere Zonendaten als `fine`-Zonendaten verwendet werden.

Die folgende Abbildung zeigt, dass eine gefundene Position auf der falschen Seite des Objekts liegt, da die falschen Zonendaten verwendet wurden.



xx0500002244

Die folgende Abbildung zeigt, dass keine Position gefunden wurde, weil die falschen Zonendaten verwendet wurden.

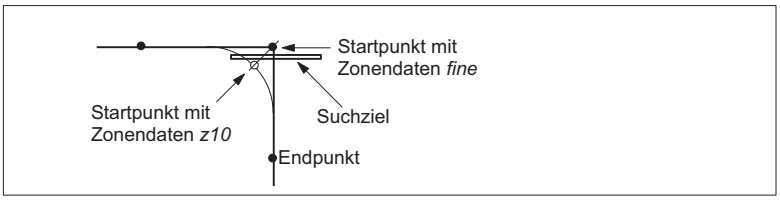


xx0500002245

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die folgende Abbildung zeigt, dass keine Position gefunden wurde, weil die falschen Zonendaten verwendet wurden.



xx0500002246

Einschränkungen für die Suche bei koordinierten Synchronbewegungen:

- Bei Verwendung von `SearchL`, `SearchC` oder `SearchExtJ` für eine Programmtask und einer anderen Bewegungsinstruktion für eine andere Programmtask kann die fliegende Suche nur mit dem Schalter `\Sup` ausgeführt werden. Außerdem kann die Fehlerbehandlung nur mit `TRYNEXT` ausgeführt werden.
- Es können alle Suchfunktionen verwendet werden, wenn einige der Instruktionen `SearchL`, `SearchC` oder `SearchExtJ` in allen beteiligten Programmtasks mit koordinierten Synchronbewegungen verwendet und Suchergebnisse aus demselben digitalen Eingangssignal generiert werden. Hierdurch werden in allen Suchinstruktionen Suchergebnisse synchron generiert. Außerdem muss die Fehlerbehandlung in allen Programmtasks identisch sein.

Während der Ausführung einer Suche ist das Speichern der aktuellen Bahn mit der Instruktion `StorePath` nicht zulässig.

Wiederholgenauigkeit für eine gefundene Position mit TCP-Geschwindigkeit 20-1000 mm/s beträgt 0,1-0,3 mm.

Typische Verzögerungsstrecke bei einer Suchgeschwindigkeit von 50 mm/s:

- ohne TCP auf der Bahn (Schalter `\Stop`) 1-3 mm
- mit TCP auf der Bahn (Schalter `\PStop`) 15-25 mm
- mit TCP auf der Bahn (Schalter `\SStop`) 4-8 mm

Einschränkungen beim Suchen eines Förderers:

- Bei einer Suche wird der Roboter angehalten, wenn das Objekt gefunden wird oder die Suche fehlschlägt. Führen Sie daher die Suche in der Bewegungsrichtung des Förderers aus, und fahren Sie nach dem Anhalten der Suche mit einer Bewegung an eine sichere Position fort. Verwenden Sie die Fehlerbehandlung, um den Roboter beim Fehlschlagen der Suche an eine sichere Position zu bewegen.
- Die Wiederholgenauigkeit für die Trefferposition der Suche ist bei der Suche für einen Förderer geringer und hängt von der Geschwindigkeit des Förderers sowie der Gleichmäßigkeit der Geschwindigkeit ab.

Syntax

```
SearchL
  ['\' Stop ','] | ['\' PStop ','] | ['\' SStop ','] | ['\' Sup
       ',']
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.225 SearchL - Lineare Suchbewegung des Roboters

RobotWare - OS

Fortsetzung

```
[Signal ':='] <variable (VAR) of signaldi> |
[PersBool ':='] <persistent (PERS) of bool>
['\' Flanks] |
['\' PosFlank] |
['\' NegFlank] |
['\' HighLevel] |
['\' LowLevel] ','
[SearchPoint ':=' ] <var or pers (INOUT) of robtarget>','
[ToPoint ':='] <expression (IN) of robtarget>
['\' ID ':=' <expression (IN) of identno>]','
[Speed ':=' ] <expression (IN) of speeddata>
['\' V ':=' <expression (IN) of num> ] |
['\' T ':=' <expression (IN) of num> ]','
[Tool ':=' ] <persistent (PERS) of tooldata>
['\' WObj ':=' <persistent (PERS) of wobjdata> ]
['\' Corr ]
['\' TLoad ':=' <persistent (PERS) of loaddata> ]';'
```

Weitere Informationen

Information über	Siehe
Kreisförmige Suchvorgänge	*SearchC - Kreisförmige Suchbewegung des Roboters auf Seite 661*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lineare Bewegung des Roboters	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Lineare Bewegung	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Verwenden von Fehlerbehandlungsroutinen	*Technisches Referenzhandbuch - RAPID Overview*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.226 SenDevice - Verbinden mit einer Sensoreinheit

Verwendung

`SenDevice` wird für den Anschluss an einen mit der seriellen Sensorschnittstelle verbundenen Sensor verwendet.

Die Sensorschnittstelle kommuniziert mit Sensoren über E/A-Geräte.

Konfigurationsbeispiel

Dies ist ein Beispiel für eine Sensorkanalkonfiguration.

Diese Parameter gehören zum Typ *Transmission Protocol* in der Parametergruppe *Communication*.

Name	Type	Remote Address	Remote Port
sen1:	SOCKDEV	192.168.125.101	6344

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SenDevice`:

Beispiel 1:

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
VAR pos SensorPos;
! Connect to the sensor device“ sen1:” (defined in sio.cfg).
SenDevice "sen1:";
! Request start of sensor meassurements
WriteVar "sen1:", SensorOn, 1;
! Read a cartesian position from the sensor.
SensorPos.x := ReadVar "sen1:", XCoord;
SensorPos.y := ReadVar "sen1:", YCoord;
SensorPos.z := ReadVar "sen1:", ZCoord;
! Stop sensor
WriteVar "sen1:", SensorOn, 0;
```

Argumente

```
SenDevice device
```

`device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

Syntax

```
ReadBlock
  [device ':='] <expression(IN) of string>','
  [BlockNo ':='] <expression (IN) of num>','
  [FileName ':='] <expression (IN) of string>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.226 SenDevice - Verbinden mit einer Sensoreinheit

Sensor Interface

Fortsetzung

Weitere Informationen

Information über	Siehe
Schreiben einer Sensorvariablen	*WriteVar – Variable schreiben auf Seite 1132*
Lesen einer Sensorvariablen	*ReadVar - Lesen einer Variablen von einem Gerät auf Seite 1474*
Schreiben eines Sensordatenblocks	*WriteBlock – Datenblock auf Gerät schreiben auf Seite 1121*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.227 Set - Setzen eines digitalen Ausgangssignals

Verwendung

`Set` wird verwendet, um den Wert eines digitalen Ausgangssignals auf eins zu setzen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Set`:

Beispiel 1:

```
Set do15;
```

Das Signal `do15` wird auf 1 gesetzt.

Beispiel 2:

```
Set weldon;
```

Das Signal `weldon` wird auf 1 gesetzt.

Argumente

```
Set Signal
```

`Signal`

Datentyp: `signaldo`

Der Name des Signals, das auf eins gesetzt werden soll.

Programmabarbeitung

Es gibt eine kurze Verzögerung, bevor das Signal physisch seinen neuen Wert erhält. Wenn die Programmabarbeitung erst fortfahren soll, nachdem das Signal den neuen Wert aufweist, verwenden Sie die Instruktion `SetDO` mit dem optionalen Parameter `\Sync`.

Der wahre Wert hängt von der Konfiguration des Signals ab. Wenn das Signal in den Systemparametern invertiert wird, wird der physische Kanal durch diese Instruktion auf null gesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.227 Set - Setzen eines digitalen Ausgangssignals

RobotWare - OS

Fortsetzung

Syntax

```
Set
  [ Signal ':=' ] < variable (VAR) of signaldo > ';'
```

Weitere Informationen

Information über	Siehe
Setzen eines digitalen Ausgangssignals auf null	*Reset - Setzt ein digitales Ausgangssignal zurück auf Seite 611*
Ändern des Werts eines digitalen Ausgangssignals	*SetDO - Ändern des Werts eines digitalen Ausgangssignals auf Seite 710*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.228 SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge

Verwendung

`SetAllDataVal`(*Set All Data Value*) ermöglicht es, einen neuen Wert für alle Datenobjekte eines bestimmten Typs zu setzen, die einer vorgegebenen Grammatik entsprechen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SetAllDataVal`:

```
VAR mydata mydata0:=0;
...
SetAllDataVal "mydata"\TypeMod:="mytypes"\Hidden,mydata0;
```

Damit werden alle Datenobjekte vom Datentyp `mydata` im System auf den gleichen Wert wie die Variable `mydata0` gesetzt (in diesem Beispiel auf `0`). Der benutzerdefinierte Datentyp `mydata` ist im Modul `mytypes` definiert.

Argumente

```
SetAllDataVal Type [\TypeMod] [\Object] [\Hidden] Value
```

`Type`

Datentyp: `string`

Der Typname des zu setzenden Datenobjekts

`[ \TypeMod ]]`

Type Module

Datentyp: `string`

Der Modulname, in dem der Datentyp bei der Verwendung anwenderdefinierter Datentypen definiert ist

Das Argument `TypeMode` kann nicht für Daten in Modulen verwendet werden, die als `-Shared` oder `-Installed` installiert sind. Der Modulname ist für diese Daten nicht verfügbar.

Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *Automatic Loading of Modules*.

`[ \Object ]`

Datentyp: `string`

Das Standardverhalten ist das Setzen aller Datenobjekte des obigen Datentyps. Durch diese Option ist es möglich, eines oder mehrere Objekte mit einem regulären Ausdruck zu benennen. Siehe auch die Instruktion `SetDataSearch`.

`[ \Hidden ]`

Datentyp: `switch`

Das trifft auch auf Datenobjekte zu, die sich in Routinen (Routinedaten oder -parameter) befinden oder durch eine Routine in der Aufrufkette verdeckt sind.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.228 SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge

RobotWare - OS

Fortsetzung

`Value`

Datentyp: `anytype`

Dieses Argument hält den neuen zu setzenden Wert Der Datentyp muss mit dem Datentyp für das zu setzende Objekt übereinstimmen.

Programmabarbeitung

Die Instruktion schlägt fehl, wenn die Spezifikation für `Type` oder `TypeMod` falsch ist.

Wenn das übereinstimmende Datenobjekt ein Datenfeld ist, werden alle Elemente in dem Datenfeld auf den angegebenen Wert gesetzt.

Wenn es sich bei dem übereinstimmenden Datenobjekt um schreibgeschützte Daten handelt, wird der Wert nicht geändert.

Wenn das System keine übereinstimmenden Datenobjekte enthält, wird dies von der Instruktion akzeptiert und sie kehrt erfolgreich zurück.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_SYMBOL_TYPE`	Das Datenobjekt und die im Argument `Value` verwendete Variable weisen unterschiedliche Typen auf. Wenn `ALIAS`-Datentypen verwendet werden, erhalten Sie ebenfalls diesen FEHLER, auch wenn die Typen den gleichen Basisdatentyp aufweisen.

Einschränkungen

Bei dem Datentyp „semivalue“ ist eine Suche nach dem zugehörigen „value“-Datentyp nicht möglich. Zum Beispiel werden bei der Suche nach `dionum` keine Suchergebnisse für das Signal `signaldi` zurückgegeben, und bei der Suche nach `num` werden keine Suchergebnisse für die Signale `signalgi` oder `signalai` zurückgegeben.

Ein Wert kann nicht auf eine Variable gesetzt werden, die in einem integrierten RAPID-Modul als `LOCAL` deklariert ist.

Syntax

```
SetAllDataVal
  [Type ':='] <expression (IN) of string>
  ['\'TypeMod ':='<expression (IN) of string>]
  ['\'Object ':='<expression (IN) of string>]
  ['\'Hidden ] ','
  [Value ':='] <variable (VAR) of anytype>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Definieren eines Symbols, das in einem Suchvorgang gesetzt wird	*SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird auf Seite 702*
Abrufen des nächsten passenden Symbols	*GetNextSym - Abrufen des nächsten passenden Symbols auf Seite 1328*
Abrufen des Werts eines Datenobjekts	*GetDataVal - Abrufen des Werts eines Datenobjekts auf Seite 228*
Setzen des Werts eines Datenobjekts	*SetDataVal - Setzen des Werts eines Datenobjekts auf Seite 707*
Zugehöriger Datentyp "datapos"	*datapos – Umschließender Block für ein Datenobjekt auf Seite 1687*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.229 SetAO - Ändern des Werts eines analogen Ausgangssignals

Verwendung

`SetAO` wird verwendet, um den Wert eines analogen Ausgangssignals zu ändern.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SetAO`:

Siehe auch *Weitere Beispiele auf Seite 701*.

Beispiel 1:

```
SetAO ao2, 5.5;
```

Das Signal `ao2` wird auf `5.5.` gesetzt.

Argumente

```
SetAO Signal Value
```

`Signal`

Datentyp: `signalao`

Der Name des analogen Ausgangssignals, das geändert werden soll.

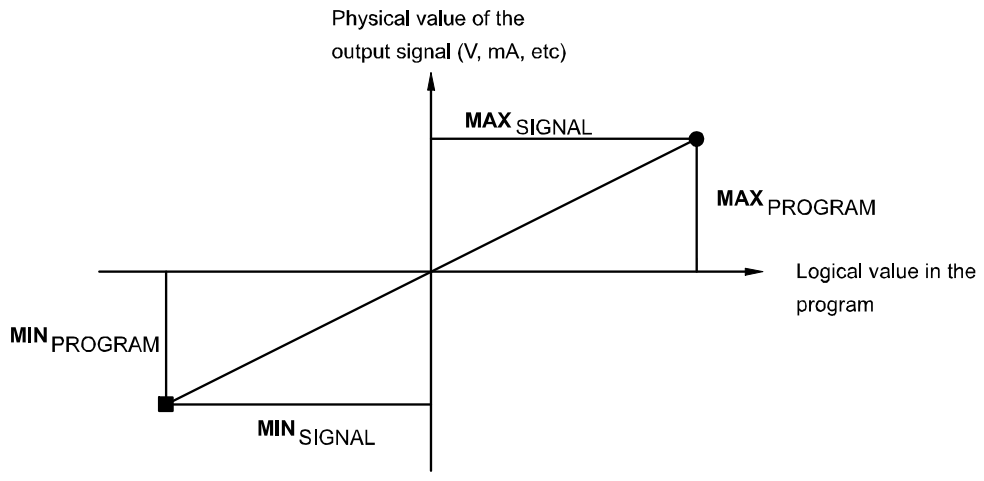
`Value`

Datentyp: `num`

Der gewünschte Wert des Signals.

Programmabarbeitung

Der programmierte Wert wird skaliert (in Übereinstimmung mit den Systemparametern), bevor er an den physischen Kanal gesendet wird In der Abbildung unten wird ein Diagramm über die Skalierung von analogen Signalwerten gezeigt.



xx0500002408

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte Value-Argument für das angegebene analoge Ausgangssignal Signal liegt außerhalb des zulässigen Bereichs.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Weitere Beispiele

Weitere Beispiele für die Instruktion SetAO sind unten aufgeführt.

Beispiel 1:

```
SetAO weldcurr, curr_outp;
```

Das Signal weldcurr wird auf denselben Wert gesetzt wie der aktuelle Wert der Variable curr_outp.

Syntax

```
SetAO
  [ Signal ':=' ] < variable (VAR) of signalao > ','
  [ Value ':=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.230 SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird

Verwendung

`SetDataSearch` wird zusammen mit der Funktion `GetNextSym` verwendet, um Datenobjekte aus dem System abzurufen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SetDataSearch`:

Beispiel 1:

```
VAR datapos block;
VAR string name;
...
SetDataSearch "robtarget"\InTask;
WHILE GetNextSym(name,block \Recursive) DO
...
```

Diese Sitzung findet alle Objekte für ein `robtarget` in der Task.

Beispiel 2:

```
RECORD testrecord
  num value1;
  num value2;
ENDRECORD

VAR datapos block;
VAR string name;
VAR testrecord mydata1:= [1,2];
VAR testrecord mydata2:= [3,4];
...
SetDataSearch "testrecord" \TypeMod:="MYMODULE" \InTask;
WHILE GetNextSym(name,block \Recursive) DO
...
```

Der Datentyp `testrecord` ist ein benutzerdefinierter Datentyp, der im Modul `MYMODULE` definiert ist. Diese Sitzung findet alle Objekte für einen `testrecord` in der Task.

Argumente

```
SetDataSearch Type [\TypeMod] [\Object] [\PersSym]
     [\VarSym][\ConstSym] [\InTask] | [\InMod]
     [\InRout][\GlobalSym] | [\LocalSym]
```

`Type`

Datentyp: `string`

Der Name des Datentyps des Datenobjekts, das abgerufen werden soll

`[ \TypeMod ]]`

Type Module

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Modulname, in dem der Datentyp bei der Verwendung anwenderdefinierter Datentypen definiert ist

Das Argument `TypeMode` kann nicht für Daten in Modulen verwendet werden, die als `-Shared` oder `-Installed` installiert sind. Der Modulname ist für diese Daten nicht verfügbar.

Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *Automatic Loading of Modules*.

`[ \Object ]]`

Datentyp: `string`

Das Standardverhalten ist das Setzen aller Datenobjekte des obigen Datentyps. Durch diese Option ist es möglich, eines oder mehrere Objekte mit einem regulären Ausdruck zu benennen.

Ein regulärer Ausdruck ist ein hervorragendes Mittel, um die Grammatik anzugeben, die den Datenobjektnamen entsprechen soll. Die Zeichenfolge kann aus gewöhnlichen Zeichen und Metazeichen bestehen. Ein Metazeichen ist ein spezieller Operator, der für die Darstellung einer oder mehrerer gewöhnlicher Zeichen in einer Zeichenfolge zur Sucherweiterung verwendet wird. Es kann festgestellt werden, ob eine Zeichenfolge als Ganzes oder eine Teilzeichenfolge einem bestimmten Muster entspricht.

In einem regulären Ausdruck treffen alphanumerische Zeichen auf sich selbst zu. Das Muster „abc“ entspricht also nur einem Datenobjekt mit dem Namen „abc“. Um eine Übereinstimmung mit allen Datenobjektnamen zu erzielen, die die Zeichenfolge „abc“ enthalten, müssen bestimmte Metazeichen hinzugefügt werden. Der gewöhnliche Ausdruck dafür lautet „.*abc.*“.

Der verfügbare Satz von Metazeichen ist unten dargestellt.

Ausdruck	Bedeutung
.	Any single character.
[s]	Ein einzelnes Zeichen im nicht leeren Satz s, wobei s eine Zeichenfolge ist. Bereiche können als c-c angegeben werden.
[^s]	Ein beliebiges, einzelnes Zeichen außerhalb des Satzes s.
r*	Null oder mehr Vorkommen eines regulären Ausdrucks r.
r+	Ein oder mehr Vorkommen eines regulären Ausdrucks r.
r?	Null oder ein Vorkommen eines regulären Ausdrucks r.
(r)	Der reguläre Ausdruck r. Wird verwendet, um die regulären Ausdrücke voneinander zu trennen.
r | r'	Die regulären Ausdrücke r oder r'.
.*	Eine beliebige Zeichenabfolge (null, ein oder mehrere Zeichen).

Das Standardverhalten ist das Akzeptieren aller Symbole. Wenn jedoch eines oder mehrere der folgenden `PersSym`, `VarSym` oder `ConstSym` angegeben ist, dann startet die Suche auf Systemebene.

`[ \PersSym ]`

Persistent Symbols

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.230 SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird
RobotWare - OS
Fortsetzung

Datentyp: `switch`

Persistente Variablensymbole (`PERS`) werden akzeptiert.

`[ \VarSym ]`

Variable Symbols

Datentyp: `switch`

Variablensymbole (`VAR`) werden akzeptiert.

`[ \ConstSym ]`

Constant Symbols

Datentyp: `switch`

Konstantensymbole (`CONST`) werden akzeptiert.

Wenn nicht eines der Flags `\InTask` oder `\InMod` angegeben ist, dann startet die Suche auf Systemebene. Die Systemebene ist die Wurzel für alle anderen Symboldefinitionen im Symbolbaum. Auf der Systemebene befinden sich alle integrierten Symbole sowie jene auf Taskebene. Auf der Taskebene befinden sich neben allen geladenen, globalen Symbole, auch jene auf Modulebene.

Wenn das Flag `\Recursive` in`GetNextSym` gesetzt ist, wird die Suche in allen geladenen Modulen und Routinen unterhalb der Systemebene durchgeführt.

`[ \InTask ]`

In Task

Datentyp: `switch`

Die Suche auf Taskebene starten. Auf der Taskebene befinden sich alle geladenen, globalen Symbole und die Handhabung für die Modulebene.

Wenn das Flag `\Recursive` in `GetNextSym` gesetzt ist, wird die Suche in allen geladenen Modulen und Routinen unterhalb der Taskebene durchgeführt.

`[ \InMod ]`

In Module

Datentyp: `string`

Die Suche auf der angegebenen Modulebene starten. Auf der Modulebene befinden sich alle geladenen, globalen und lokalen Symbole, die in dem angegebenen Modul deklariert sind, und die Handhabung für die Routinenebenen.

Wenn das Flag `\Recursive` in `GetNextSym` gesetzt ist, wird die Suche in allen geladenen Routinen unterhalb der angegebenen Modulebene durchgeführt (deklariert im angegebenen Modul).

`[ \InRout ]`

In Routine

Datentyp: `string`

Suche nur auf der angegebenen Routinenebene.

Der Modulname für die Routine muss im Argument `\InMod.` angegeben werden.

Das Standardverhalten ist der Abgleich der lokalen sowie der globalen Modulsymbole. Wenn jedoch eines der folgenden `\GlobalSym` or `\LocalSym`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

angegeben ist, dann werden nur jene Symbole akzeptiert, die der Spezifikation entsprechen.

`[ \GlobalSym ]`

Global Symbols

Datentyp: `switch`

Lokale Modulsymbole überspringen.

`[ \LocalSym ]`

Local Symbols

Datentyp: `switch`

Globale Modulsymbole überspringen.

Programmabarbeitung

Die Instruktion schlägt fehl, wenn die Spezifikation für `Type`, `TypeMod`, `InMod` oder `InRout` falsch ist.

Wenn das System keine übereinstimmenden Objekte enthält, wird dies von der Anweisung akzeptiert und erfolgreich zurückgegeben, wobei `GetNextSym` jedoch zuerst `FALSE` zurückgibt.

Einschränkungen

Datenfeldobjekte können in der Symbolsuche nicht definiert und in einer Suchsequenz nicht gefunden werden.

Bei dem Datentyp „semivalue“ ist eine Suche nach dem zugehörigen „value“-Datentyp nicht möglich. Zum Beispiel werden bei der Suche nach `dionum` keine Suchergebnisse für das Signal `signaldi` zurückgegeben, und bei der Suche nach `num` werden keine Suchergebnisse für die Signale `signalgi` oder `signalai` zurückgegeben.

Installierte integrierte Symbole, die als `LOCAL` deklariert sind, werden niemals gefunden, unabhängig davon, ob das Argument `\GlobalSym`, `\LocalSym` oder keines dieser Argumente verwendet wird.

Installierte integrierte Symbole, die als global oder als `TASK` deklariert sind, werden niemals gefunden, unabhängig davon, ob das Argument `\GlobalSym`, `\LocalSym` oder keines dieser Argumente verwendet wird.

`SetDataSearch`kann nicht zum Suchen von Daten des mit `ALIAS`-Code definierten `RAPID`-Datentyps verwendet werden. Für den vordefinierten `ALIAS`-Datentyp gibt es keine Begrenzung.

Syntax

```
SetDataSearch
  [Type ':='] <expression (IN) of string>
  ['\'TypeMod ':='<expression (IN) of string>]
  ['\'Object ':='<expression (IN) of string>]
  ['\'PersSym]
  ['\'VarSym]
  ['\'ConstSym]
  ['\'InTask]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.230 SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird

RobotWare - OS

Fortsetzung

```
|['\'InMod' :='<expression (IN) of string>]
['\'InRout ':='<expression (IN) of string>]
['\'GlobalSym]
|['\'LocalSym]';'
```

Weitere Informationen

Information über	Siehe
Abrufen des nächsten passenden Symbols	*GetNextSym - Abrufen des nächsten passenden Symbols auf Seite 1328*
Abrufen des Werts eines Datenobjekts	*GetDataVal - Abrufen des Werts eines Datenobjekts auf Seite 228*
Setzen des Werts vieler Datenobjekte	*SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge auf Seite 697*
Zugehöriger Datentyp `datapos`	*datapos – Umschließender Block für ein Datenobjekt auf Seite 1687*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.231 SetDataVal - Setzen des Werts eines Datenobjekts

Verwendung

`SetDataVal` (*Set Data Value*) ermöglicht das Setzen eines Werts für ein Datenobjekt, das mit einer Zeichenfolgenvariablen angegeben ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SetDataVal`:

Beispiel 1:

```
VAR datapos block;
VAR bool truevar:=TRUE;
...
SetDataSearch "bool" \Object:="my.*" \InMod:="mymod"\LocalSym;
WHILE GetNextSym(name,block) DO
  SetDataVal name\Block:=block,truevar;
ENDWHILE
```

Dieser Vorgang setzt alle lokalen Datenobjekte des Typs `bool`, die im Modul `mymod` mit `my` beginnen, auf `TRUE`.

Beispiel 2:

```
VAR string StringArrVar_copy{2};
...
StringArrVar_copy{1} := "test1";
StringArrVar_copy{2} := "test2";
SetDataVal "StringArrVar", StringArrVar_copy;
```

Nach diesem Vorgang enthält das Datenfeld `StringArrVar` die beiden Zeichenfolgen `test1` und `test2`.

Argumente

```
SetDataVal Object [\Block]|[\TaskRef]|[\TaskName] Value
```

`Object`

Datentyp: `string`

Name des Datenobjekts

`[ \Block ]`

Datentyp: `datapos`

Der eingeschlossene Block für das Datenobjekt. Kann nur mit der Funktion `GetNextSym` abgerufen werden.

Wenn dieses Argument weggelassen wird, wird der Wert des sichtbaren Datenobjekts im Gültigkeitsbereich der aktuellen Programmabarbeitung gesetzt.

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, in der das angegebene Datenobjekt gesucht werden soll. Bei Verwendung dieses Arguments können Sie die `PERS`- oder

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.231 SetDataVal - Setzen des Werts eines Datenobjekts
RobotWare - OS
Fortsetzung

TASKPERS-Deklaration in anderen Tasks suchen. Jede andere Deklaration führt zu einem Fehler.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps taskid zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“ (die Variablenidentität für die Task T_ROB1 lautet z. B. T_ROB1Id).

[\TaskName]

Datentyp: string

Der Name der Programmtask, in der das angegebene Datenobjekt gesucht werden soll. Bei Verwendung dieses Arguments können Sie die PERS- oder TASKPERS-Deklaration in anderen Tasks suchen. Jede andere Deklaration führt zu einem Fehler.

Value

Datentyp: anytype

Variable, die den neuen, zu setzenden Wert hält. Der Datentyp muss mit dem des zu setzenden Datenobjekts identisch sein. Der gesetzte Wert muss von einer Variablen abgerufen werden, kann jedoch in einer Variablen oder Persistenten gespeichert werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SYM_ACCESS	• Das Datenobjekt existiert nicht. • Das Datenobjekt besteht aus Read-Only-Daten. • Das Datenobjekt besteht aus Routinedaten oder Routineparametern und befindet sich nicht in der aktuellen, aktiven Routine. • Es wird in anderen Tasks nach anderen Deklarationen als PERS oder TASKPERS gesucht. [i]
ERR_INVDIM	Das Datenobjekt und die im Argument Value verwendete Variable weisen unterschiedliche Dimensionen auf.
ERR_SYMBOL_TYPE	Das Datenobjekt und die im Argument Value verwendete Variable weisen unterschiedliche Typen auf. Wenn ALIAS-Datentypen verwendet werden, erhalten Sie ebenfalls diesen FEHLER, auch wenn die Typen den gleichen Basisdatentyp aufweisen.
ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.

[i] Bei Verwendung der Argumente TaskRef oder TaskName können Sie die PERS- oder TASKPERS-Deklarationen in anderen Tasks suchen. Andere Deklarationen führen zu einem Fehler.
Das Suchen nach einer PERS, die als LOCAL deklariert ist, in anderen Tasks führt ebenfalls zu einem Fehler.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Bei dem Datentyp semivalue ist eine Suche nach dem zugehörigen value-Datentyp nicht möglich. Wenn z. B. nach `dionum` gesucht wird, werden keine Treffer für `signaldi`-Signale erzielt, und bei der Suche nach `num` werden keine Treffer für `signalgi`-Signale oder `signalai`-Signale erzielt.

Ein Wert kann nicht auf eine Variable gesetzt werden, die in einem integrierten RAPID-Modul als `LOCAL` deklariert ist.

Syntax

```
SetDataVal
  [ Object ':=' ] < expression (IN) of string >
  ['\'Block' :='<variable (VAR) of datapos>]
  |[ '\'TaskRef' :=' <variable (VAR) of taskid>]
  |[ '\'TaskName' :=' <expression (IN) of string>] ',']
  [ Value ':=' ] <variable (VAR) of anytype>]';'
```

Weitere Informationen

Information über	Siehe
Definieren eines Symbols, das in einem Suchvorgang gesetzt wird	*SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird auf Seite 702*
Abrufen des nächsten passenden Symbols	*GetNextSym - Abrufen des nächsten passenden Symbols auf Seite 1328*
Abrufen des Werts eines Datenobjekts	*GetDataVal - Abrufen des Werts eines Datenobjekts auf Seite 228*
Setzen des Werts vieler Datenobjekte	*SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge auf Seite 697*
Zugehöriger Datentyp `datapos`	*datapos – Umschließender Block für ein Datenobjekt auf Seite 1687*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.232 SetDO - Ändern des Werts eines digitalen Ausgangssignals

Verwendung

`SetDO` wird verwendet, um den Wert eines digitalen Ausgangssignals mit oder ohne Zeitverzögerung oder Synchronisierung zu ändern.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SetDO`.

Beispiel 1:

```
SetDO do15, 1;
```

Das Signal `do15` wird auf `1` gesetzt.

Beispiel 2:

```
SetDO weld, off;
```

Das Signal `weld` wird auf `off` gesetzt.

.

Beispiel 3

```
SetDO \SDelay := 0.2, weld, high;
```

Das Signal `weld` wird mit einer Verzögerung von `high` s auf `0.2` gesetzt. Die Programmabarbeitung fährt jedoch mit der nächsten Instruktion fort.

Beispiel 4:

```
SetDO \Sync ,do1, 0;
```

Das Signal `do1` wird auf `0` gesetzt. Die Programmabarbeitung wartet, bis das Signal physisch auf den angegebenen Wert gesetzt ist.

Argumente

```
SetDO [ \SDelay ]|[ \Sync ] Signal Value
```

`[ \SDelay ]`

Signal Delay

Datentyp: `num`

Verzögert die Änderung für die in Sekunden angegebene Zeitdauer (max. 2000 s). Die Programmabarbeitung fährt direkt mit der nächsten Instruktion fort. Nach der angegebenen Verzögerung wird das Signal geändert, ohne die übrige Programmabarbeitung zu beeinflussen.

`[ \Sync ]`

Synchronization

Datentyp: `switch`

Wenn dieses Argument verwendet wird, wartet die Programmabarbeitung, bis das Signal physisch auf den angegebenen Wert gesetzt ist.

`Signal`

Datentyp: `signaldo`

Der Name des zu ändernden Signals.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Value`

Datentyp: `dionum`

Der gewünschte Signalwert (0 oder 1).

Angegebener Wert	Setzt digitalen Ausgang auf
0	0
Beliebiger Wert außer 0	1

Programmabarbeitung

Der wahre Wert hängt von der Konfiguration des Signals ab. Wenn das Signal in den Systemparametern invertiert ist, hat der physische Kanal den entgegengesetzten Wert.

Wenn weder das Argument `\SDelay` noch das Argument `\Sync` verwendet wird, wird das Signal so schnell wie möglich gesetzt und die nächste Instruktion wird sofort abgearbeitet, ohne auf das physische Setzen des Signals zu warten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ARGVALERR`	Der Wert für das `SDelay`-Argument überschreitet den maximal zulässigen Wert (2000 s).
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

Wenn auf `SetDO` mit dem Argument `\SDelay` ein neues `SetDO` für das gleiche Signal folgt (mit oder ohne `\SDelay`-Argument), wird das erste `SetDO` abgebrochen, falls das zweite `SetDO` vor Ablauf der Zeitverzögerung des ersten `SetDO` ausgeführt wird.

Syntax

```
SetDO
  ['\' SDelay ':=' <expression (IN) of num>',']
  |['\'Sync',']
  [Signal ':='] <variable (VAR) of signaldo>','
  [Value ':='] <expression (IN) of dionum>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.232 SetDO - Ändern des Werts eines digitalen Ausgangssignals

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.233 SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen

Verwendung

`SetGO` wird verwendet, um den Wert einer Gruppe von digitalen Ausgangssignalen mit oder ohne Zeitverzögerung zu ändern.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SetGO`:

Beispiel 1:

```
SetGO go2, 12;
```

Das Signal `go2` wird auf `12` gesetzt. Wenn `go2` 4 Signale umfasst, z. B. Ausgang 6-9, werden die Ausgänge 6 und 7 auf null und die Ausgänge 8 und 9 auf eins gesetzt.

Beispiel 2:

```
SetGO \SDelay := 0.4, go2, 10;
```

Das Signal `go2` wird auf `10` gesetzt. Wenn `go2` 4 Signale umfasst, z. B. Ausgänge 6-9, dann sind die Ausgänge 6 und 8 auf null gesetzt, während die Ausgänge 7 und 9 mit einer Verzögerung `0.4` s auf eins gesetzt sind. Die Programmabarbeitung wird jedoch mit der nächsten Instruktion fortgesetzt.

Beispiel 3

SetGO go32, 4294967295;

Das Signal `go32` wird auf `4294967295` gesetzt. `go32` umfasst 32 Signale, die alle auf eins gesetzt sind.

Argumente

```
SetGO [ \SDelay ] Signal Value | Dvalue
```

`[ \SDelay ]`

Signal Delay

Datentyp: `num`

Verzögert die Änderung für die in Sekunden angegebene Zeitdauer (max. 2000 s). Die Programmabarbeitung fährt direkt mit der nächsten Instruktion fort. Nach der angegebenen Verzögerung wird der Wert des Signals geändert, ohne die übrige Programmabarbeitung zu beeinflussen.

Wenn das Argument ausgelassen wird, werden die Signalwerte direkt geändert.

`Signal`

Datentyp: `signalgo`

Der Name der zu ändernden Signalgruppe.

`Value`

Datentyp: `num`

Der gewünschte Wert der Signalgruppe (eine positive Ganzzahl) wird in der folgenden Tabelle angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.233 SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen
RobotWare - OS
Fortsetzung

Der zulässige Wert hängt von der Anzahl der Signale in der Gruppe ab. Der Datentyp num kann den Wert für eine Gruppe von höchstens 23 Signalen enthalten.

Dvalue

Datentyp: dnum

Der gewünschte Wert der Signalgruppe (eine positive Ganzzahl) wird in der folgenden Tabelle angezeigt.

Der zulässige Wert hängt von der Anzahl der Signale in der Gruppe ab. Der Datentyp dnum kann den Wert für eine Gruppe von bis zu 32 Signalen enthalten.

Anzahl der Signale	Zulässiger Value	Zulässiger Dvalue
1	0-1	0-1
2	0-3	0-3
3	0-7	0-7
4	0-15	0-15
5	0-31	0-31
6	0-63	0-63
7	0-127	0-127
8	0-255	0-255
9	0-511	0-511
10	0-1023	0-1023
11	0-2047	0-2047
12	0-4095	0-4095
13	0-8191	0-8191
14	0-16383	0-16383
15	0-32767	0-32767
16	0-65535	0-65535
17	0-131071	0-131071
18	0-262143	0-262143
19	0-524287	0-524287
20	0-1048575	0-1048575
21	0-2097151	0-2097151
22	0-4194303	0-4194303
23	0-8388607	0-8388607
24	*	0-16777215
25	*	0-33554431
26	*	0-67108863
27	*	0-134217727
28	*	0-268435455
29	*	0-536870911
30	*	0-1073741823

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Anzahl der Signale	Zulässiger Value	Zulässiger Dvalue
31	*	0-2147483647
32	*	0-4294967295

*) Das Argument `Value` vom Typ `num` kann nur bis zu 23 Signale enthalten, während das Argument `Dvalue` vom Typ `dnum` bis zu 32 Signale enthalten kann.

Programmabarbeitung

Der programmierte Wert wird in eine binäre Zahl ohne Vorzeichen konvertiert. Die binäre Zahl wird an die Signalgruppe gesendet und setzt einzelne Signale der Gruppe auf 0 bzw. 1. Aufgrund interner Verzögerungen kann der Wert des Signals kurzzeitig undefiniert sein.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ARGVALERR`	Der Wert für das `SDelay`-Argument überschreitet den maximal zulässigen Wert (2000 s).
`ERR_GO_LIM`	Das programmierte `Value`- oder `Dvalue`-Argument für das angegebene digitale Gruppenausgangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Einschränkungen

Die maximale Anzahl an Signalen für eine Gruppe beträgt beim Argument `Value` 23 und beim Argument `Dvalue` 32. Die Beschränkung gilt für alle Instruktionen und Funktionen, die Gruppensignale verwenden.

Syntax

```
SetGO
  [ '\' SDelay ':=' < expression (IN) of num > ',' ]
  [ Signal ':=' ] < variable (VAR) of signalgo > ','
  [ Value ':=' ] < expression (IN) of num >
  | [ Dvalue' :=' ] < expression (IN) of dnum > ';'
```

Weitere Informationen

Information über	Siehe
Andere E/A-Instruktionen	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.233 SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen

RobotWare - OS

Fortsetzung

Information über	Siehe
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A (Systemparameter)	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.234 SetLeadThrough - Durchführung aktivieren und deaktivieren

Verwendung

`SetLeadThrough` wird verwendet, um die Durchführung für einen TCP-Roboter zu aktivieren und zu deaktivieren.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SetLeadThrough`.

Beispiel 1:

```
SetLeadThrough \On;
```

Aktiviert die Durchführung für den TCP-Roboter `ROB_L`, wenn die Abarbeitung in der `T_ROB_L` RAPID-Task erfolgt. Als Standard wird eine `StopMove`-Instruktion erteilt, wenn die Durchführung aktiviert wird.

Beispiel 2:

```
SetLeadThrough \Off;
```

Deaktiviert die Durchführung für den TCP-Roboter `ROB_L`, wenn die Abarbeitung in der `T_ROB_L` RAPID-Task erfolgt. Als Standard wird eine `ClearPath`-Instruktion erteilt und zudem wird eine `StartMove`-Instruktion abgearbeitet.

Beispiel 3

```
SetLeadThrough \On \NoStopMove;
..
StopMove;
..
SetLeadThrough \Off \NoStartMove \NoClearPath;
..
StartMove;
```

Legen Sie die Durchführung für den TCP-Roboter fest. Die Durchführung wird erst aktiviert, wenn eine `StopMove`-Instruktion abgearbeitet oder die Programmausführung angehalten wurde. Die Deaktivierung der Durchführung erfolgt und später wird die Bewegung neu gestartet.

Argumente

```
SetLeadThrough [\On] | [\Off] [\NoStopMove] | [\NoStartMove]
     [\NoClearPath]
```

`[\On]`

Datentyp: `switch`

Durchführung aktivieren.

`[\Off]`

Datentyp: `switch`

Durchführung deaktivieren.

`[\NoStopMove]`

Datentyp: `switch`

Kann nur gemeinsam mit dem `\On`-Switch verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.234 SetLeadThrough - Durchführung aktivieren und deaktivieren
LeadThrough
Fortsetzung

Bei Verwendung des \NoStopMove-Switches wird kein StopMove-Befehl abgearbeitet. Die Durchführung wurde konfiguriert, aber nicht aktiviert. Die Durchführung wird aktiviert, wenn die Programmausführung angehalten oder wenn eine StopMove-Instruktion abgearbeitet wird.

[\NoStartMove]

Datentyp: switch

Kann nur gemeinsam mit dem \Off-Switch verwendet werden.

Wenn diese Option verwendet wird, wird der Neustart der Bewegung des TCP-Roboters befohlen. Es wird eine StartMove-Instruktion benötigt, um die Bewegung fortzusetzen.

[\NoClearPath]

Datentyp: switch

Kann nur gemeinsam mit dem \Off-Switch verwendet werden.

Die Bahn wird nicht gelöscht, wenn die Durchführung deaktiviert wird, und der TCP-Roboter fährt auf der programmierten Bahn fort, wenn der StartMove-Befehl abgearbeitet wird.

Programmabarbeitung

Der Status der Durchführung wird festgelegt, wenn das Argument \On (oder kein Argument) verwendet wird, aber erst aktiviert, wenn eine StopMove-Instruktion abgearbeitet oder die Programmausführung gestoppt wurde.

Standardmäßig wird eine StopMove-Instruktion abgearbeitet, wenn die Durchführung durch SetLeadThrough \On aktiviert wird. Eine ClearPath-Instruktion und eine StartMove-Instruktion wird abgearbeitet, wenn die Durchführung deaktiviert wird und nicht die Switches \NoClearPath oder \NoStartMove verwendet werden.

Wenn die SetLeadThrough-Instruktion von einer Nicht-Bewegungstask ausgeführt wird, erfolgt die Durchführungsaktivierung für den TCP-Roboter in der zugehörigen Bewegungstask. Der StartMove-Befehl muss über dieselbe Task stammen wie der StopMove-Befehl.

Die Aktivierung der Durchführung ist gültig, bis eine SetLeadThrough \Off-Instruktion ausgeführt wird.

Der Standardwert (keine Durchführung) wird automatisch eingestellt:

- bei Verwendung des Neustart-Modus **Reset RAPID** (RAPID zurücksetzen).
- beim Laden eines neuen Programms oder eines neuen Moduls.
- und bei Beginn der Programmabarbeitung vom Anfang.
- beim Bewegen des Programmzeigers auf main (primär).
- beim Bewegen des Programmzeiger zu einer Routine.
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.
- Wenn in den Zustand "Motoren aus" gewechselt wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_PATHDIST	Der Roboter ist für einen Neustart der unterbrochenen Bewegung zu weit von der Bahn entfernt (mehr als 10 mm oder 20 Grad). Der Roboter muss vor der Ausführung von RETRY näher an die Bahn bewegt werden.
ERR_STARTMOVE	Der Roboter befindet sich bei Ausführung von SetLeadThrough \Off im Haltezustand. Vor RETRY muss einige Zeit gewartet werden.
ERR_PROGSTOP	Der Roboter befindet sich bei Ausführung von SetLeadThrough \Off im Zustand Programmstopp. Vor RETRY muss einige Zeit gewartet werden.
ERR_ALRDY_MOVING	Der Roboter bewegt sich bei der Abarbeitung von SetLeadThrough \Off bereits. Vor RETRY muss einige Zeit gewartet werden.

Einschränkungen

- Um SetLeadThrough für dieselbe Bewegungstask auszuführen, ist nur eine von mehreren Nicht-Bewegungstasks gleichzeitig zulässig.
- SetLeadThrough kann nur bei TCP-Robotern verwendet werden.

Die Instruktionen SetLeadThrough können nur für YuMi-Roboter verwendet werden.

Syntax

```
SetLeadThrough
  ['\'On] | ['\'Off]
  ['\'NoStopMove] | ['\'NoStartMove]
  ['\'NoClearPath] ';'
```

Weitere Informationen

Information über	Siehe
Status der Durchführung prüfen	*IsLeadThrough - Status der Durchführung prüfen auf Seite 1376*
Stoppen von Bewegungen	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Fortfahren mit einer Bewegung	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806*
Fortfahren mit einer Bewegung	*StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung auf Seite 809*
Weitere Beispiele	*ClearPath - Löschen der aktuellen Bahn auf Seite 142*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.235 SetSysData - Setzen von Systemdaten

Verwendung

`SetSysData` aktiviert den spezifizierten Systemdatennamen des festgelegten Datentyps.

Mit dieser Instruktion kann das derzeit aktive Werkzeug, Werkobjekt, die Nutzlast oder die Gesamtlast für den Roboter in der aktuellen oder verbundenen Bewegungstask geändert werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SetSysData`:

Beispiel 1:

```
SetSysData tool5;
```

Das Werkzeug `tool5` ist aktiviert.

```
SetSysData tool0 \ObjectName := "tool6";
```

Das Werkzeug `tool6` ist aktiviert.

```
SetSysData anytool \ObjectName := "tool2";
```

Das Werkzeug `tool2` ist aktiviert.

Argumente

```
SetSysData SourceObject [\ObjectName]
```

`SourceObject`

Datentyp:`anytype`

Persistente Variable, die als aktuelle Systemdaten aktiv sein soll.

Der Datentyp dieses Arguments gibt auch den Typ von Systemdaten an, der für den Roboter in der tatsächlichen oder verbundenen Bewegungstask aktiviert werden soll.

Datentyp	Typ der Systemdaten
tooldata	Werkzeug
wobjdata	Werkobjekt
loaddata	Nutzlast/Gesamtlast

Komplette Datenfeld- oder Datensatzkomponente kann nicht verwendet werden.

`[ \ObjectName ]`

Datentyp:`string`

Wenn dieses optionale Argument spezifiziert ist, spezifiziert es den Namen des Datenobjekts, das aktiv sein soll (überschreibt den Namen, der im Argument `SourceObject` spezifiziert ist). Der Datentyp des zu aktivierenden Datenobjekts wird immer aus dem Argument `SourceObject` abgerufen.

Programmabarbeitung

Das aktuelle aktive Systemdatenobjekt für das Werkzeug, Werkstück, die Nutzlast oder die Gesamtlast wird den Argumenten entsprechend gesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beachten Sie, dass diese Instruktion nur ein neues Datenobjekt aktiviert (oder das gleiche wie zuvor) und niemals den Wert eines Datenobjekts ändert.

Syntax

```
SetSysData
  [ SourceObject':='] < persistent(PERS) of anytype>
  ['\'ObjectName':=' < expression (IN) of string> ] ';'
```

Weitere Informationen

Information über	Siehe
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition einer Nutzlast	*loaddata - Lastdaten auf Seite 1723*
Abrufen von Systemdaten	*GetSysData - Abrufen von Systemdaten auf Seite 235*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast. (Parametergruppe Controller, TypGeneral RAPID, Aktionswerte *ModalPayLoadMode*)	*Technisches Referenzhandbuch - Systemparameter*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.236 SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung

Verwendung

`SetupCyclicBool` wird verwendet, um eine logische Bedingung zu konfigurieren, die zyklisch ausgewertet und einer auf eine persistenten booleschen Variable, einer *Cyclic bool*, zugeordnet wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SetupCyclicBool`.

Siehe auch *Weitere Beispiele auf Seite 723*.

Beispiel 1:

```
PERS bool cyclicflag1;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  ...
```

Konfiguriert eine zyklische Auswertung des logischen Zustand `di1=1 AND do2=1` und weist das Ergebnis der persistenten booleschen Variable `cyclicflag1` zu.

Argumente

```
SetupCyclicBool Flag Cond [\Signal]
```

`Flag`

Datentyp: `bool`

Die persistente boolesche Variable, die den Wert der logischen Bedingung speichert.

`Cond`

Datentyp: `bool`

Der logische Ausdruck, der zyklisch ausgewertet werden soll.

Der Ausdruck kann bestehen aus:

- Konstanten oder persistenten Variablen der Typen `bool`, `num` und `dnum` (und Alias von `bool`, `num` und `dnum`).
- Globale digitale Eingangs- und Ausgangssignale
- Operanden: 'NOT' 'AND' 'OR' 'XOR' '=' '(' ')'

`[\Signal]`

Datentyp: `signaldo`

Das Ergebnis der logischen Bedingung wird auf das digitale Ausgangssignal gesetzt, das im optionalen Argument `Signal` verwendet wird, wenn die persistente boolesche Variable aktualisiert wird.

Es wird davon abgeraten, das resultierende Signal als einen Teil der Bedingung für einen Cyclic bool zu verwenden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Mit diesem Befehl ist es möglich, komplexe Bedingungen einzurichten und das Flag "Cyclic" zu verwenden, anstatt zu sehen, ob die Bedingung erfüllt ist oder nicht.

Die zyklische Auswertung der logischen Bedingung und die Zuordnung zur persistenten booleschen Variable erfolgt alle 12 ms.

Das Verhalten der Cyclic bool-Funktion ist konfigurierbar. Weitere Informationen finden Sie unter *Anwendungshandbuch - Steuerungssoftware IRC5* und *Technisches Referenzhandbuch - Systemparameter*.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID deklarierte Variable. Sie wurde nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden (gilt nur für ICI-Feldbus).

Weitere Beispiele

Weitere Beispiele für die Instruktion `SetupCyclicBool` sind unten aufgeführt.

Beispiel 1:

```
ALIAS bool aliasBool;
PERS bool cyclicflag1;
TASK PERS aliasBool cyclicflag2:=FALSE;
PERS aliasBool flag1:=FALSE;
TASK PERS aliasBool flag2:=FALSE;
CONST num HIGH:=1;
CONST num LOW:=0;

PROC main()
  SetupCyclicBool cyclicflag1, (di1=HIGH AND di2=HIGH AND di3=LOW)
       OR flag1=TRUE;
  SetupCyclicBool cyclicflag2, di4=HIGH AND flag2=TRUE;
  ...
  WaitUntil cyclicflag1=TRUE;
  IF cyclicflag2 = TRUE THEN
    MoveL p1, v1000, z30, tool2;
  ELSE
    MoveL p2, v1000, z30, tool2;
  ENDIF
  ...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.236 SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung
RobotWare - OS
Fortsetzung

Das Beispiel oben richtet eine zyklische Auswertung von zwei Ausdrücken ein. Die Ausführung wartet, bis `cyclicflag1` gesetzt ist. `cyclicflag2` bestimmt, an welche Position sich der Roboter bewegen soll.

Beispiel 2:

```
!This condition is wrong:
SetupCyclicBool m1, 5;

!This condition is correct:
SetupCyclicBool m1, myNum = 5;
```

Die erste Bedingung ist nicht korrekt, da 5 kein boolescher Wert ist. Die zweite Bedingung ist korrekt, da der Vergleich als boolesche Bedingung ausgewertet werden kann, d.h. `TRUE` oder `FALSE`.

Einschränkungen

- Der Ausdruck muss mit einem booleschen Wert `TRUE` oder `FALSE` ausgewertet werden. Alle Teile des Ausdrucks müssen auch in einem booleschen Wert `TRUE` oder `FALSE` ausgewertet werden.
- Jede `PERS num`- oder `dnum`-, `CONST num`- oder `dnum`- oder literale `num`- oder `dnum`-Wert, der in einer Bedingung verwendet wird, muss vom Typ Ganzzahl sein. Die Verwendung eines Dezimalwerts führt zu einem schwerwiegenden Fehler.

Syntax

```
SetupCyclicBool
  [Flag ':='] <persistent (PERS) of bool>','
  [Cond ':='] <expression (IN) of bool>
  ['\' Signal ':=' <variable (VAR) of signaldo>]';'
```

Weitere Informationen

Information über	Siehe
Prüfen, ob eine persistente Variable eine Cyclic bool ist	*IsCyclicBool - Prüft, ob eine persistente Variable eine Cyclic bool ist auf Seite 1369*
Entfernen einer Cyclic bool-Bedingung	*RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung auf Seite 601*
Alle Cyclic bool-Bedingungen entfernen	*RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen auf Seite 599*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.237 SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP

Verwendung

`SetupSuperv` wird verwendet, um Bedingungen für zu überwachenden E/A-Signale festzulegen. Die Bedingungen werden in verschiedenen Listen gesammelt:

- PRE
- PRE_START
- END_PRE
- START
- MAIN
- END_MAIN
- START_POST1
- POST1
- END_POST1
- START_POST2
- POST2
- END_POST2

Weitere Informationen über Überwachungslisten finden Sie unter *Application manual - Continuous Application Platform*.

Als optionaler Parameter kann ein Ausgangssignal festgelegt werden. Das Ausgangssignal wird auf hoch gesetzt, wenn die vorgegebene Bedingung nicht erfüllt wird.

Grundlegendes Beispiel

```
PROC main()
  InitSuperv;
  SetupSuperv diWR_EST, ACT, SUPERV_MAIN \ErrIndSig:= do_WR_Sup;
  SetupSuperv diGA_EST, ACT, SUPERV_MAIN;
  CapL p2, v100, cdata1, weavestart, weave, fine, tWeldGun;
ENDPROC
```

`SetupSuperv` wird verwendet, um die Überwachung von Signalen einzurichten. Wenn das Signal *diWR_EST* während der SUPERV_MAIN-Phase ausfällt, wird das digitale Ausgangssignal *do_WR_Sup* auf hoch gesetzt.

Die `SetupSuperv`-Instruktion sollte nur ausgeführt werden, wenn die Überwachungsdaten geändert werden. Wenn die Überwachungsdaten sich niemals ändern, kann für sie ein Modul angelegt werden, das beim Anlaufen ausgeführt wird.

Argumente

```
SetupSuperv Signal Condition Listtype [\ErrIndSig]
```

Signal

Datentyp: `signaldi`

Digitales Signal, das überwacht werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.237 SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP
Continuous Application Platform (CAP)
Fortsetzung

Condition

Datentyp: num

Der Name, der eine der folgenden verfügbaren Bedingungen repräsentiert:

ACT:	Wird für die Statusüberwachung verwendet. Erwarteter Signalstatus während der Überwachung: aktiv. Wenn das Signal passiv wird, löst es die Überwachung aus.
PAS:	Wird für die Statusüberwachung verwendet. Erwarteter Signalstatus während der Überwachung: passiv. Wenn das Signal aktiv wird, löst es die Überwachung aus.
POS_EDGE:	Wird für Handshake-Überwachung verwendet. Erwarteter Signalstatus am Ende der Überwachung: aktiv. Wenn das Signal innerhalb des gewählten Zeitüberschreitung nicht aktiv wird, löst es die Überwachung aus.
NEG_EDGE:	Wird für Handshake-Überwachung verwendet. Erwarteter Signalstatus am Ende der Überwachung: passiv. Wenn das Signal innerhalb des gewählten Zeitüberschreitung nicht passiv wird, löst es die Überwachung aus.

Listtype

Datentyp: num

Der Name, der die Anzahl der verschiedenen Listen (beispielsweise Phasen des Prozesses) repräsentiert:

- SUPERV_PRE
- SUPERV_PRE_START
- SUPERV_END_PRE
- SUPERV_START
- SUPERV_MAIN
- SUPERV_END_MAIN
- SUPERV_START_POST1
- SUPERV_POST1
- SUPERV_END_POST1
- SUPERV_START_POST2
- SUPERV_POST2
- SUPERV_END_POST2

[\ErrIndSig]

Datentyp: signaldo

Wird verwendet, um die Bedingung anzugeben, die nicht erfüllt war, als der Fehler aufgetreten ist. Wenn der Fehler auftritt, wird der Wert dieses Signals auf 1 gesetzt. Das ist ein optionaler Parameter.

Programmabarbeitung

Das gegebene Signal und seine Bedingung wird zur ausgewählten Liste hinzugefügt. Wenn ein Signal ausfällt, meldet die Instruktion CapL/CapC, dass während der festgelegten Phase ein Überwachungsfehler eingetreten ist und welche Signale ausgefallen sind.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehler

CAP_SPV_LIM

Die maximale Anzahl der eingerichteten Überwachungen wurde überschritten.

CAP_SPV_UNK_LST

Die Überwachungsliste ist nicht bekannt.

Einschränkungen

Nur digitale Eingangssignale können überwacht werden.

Die Statusüberwachung gilt für eine komplette Sequenz von CAP-Instruktionen (siehe Abschnitt *Supervision and process phases* in *Application manual - Continuous Application Platform*).

Syntax

```
SetupSuperv
  [Signal ':='] < variable (VAR) of signaldi > ','
  [Condition ':='] < variable (IN) of num > ','
  [Listtype ':='] < variable (IN) of num >
  [\ErrIndSig ':=' < variable (VAR) of signaldo >] ';'
```

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instruktion `InitSuperv`	*InitSuperv - Zurücksetzen der gesamten CAP-Überwachung auf Seite 284*
Instruktion `RemoveSuperv`	*RemoveSuperv Bedingung für ein Signal entfernen auf Seite 607*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.238 SiConnect - Sensorschnittstelle verbinden

Verwendung

`SiConnect` wird verwendet, um eine Verbindung zu einem externen Gerät herzustellen.

Grundlegende Beispiele

Ein grundlegendes Beispiel für die Instruktion `SiConnect` ist unten aufgeführt.

Siehe auch *Weitere Beispiele auf Seite 728*.

Beispiel 1:

```
PERS sensor AnyDevice;
...
  SiConnect AnyDevice;
```

Herstellung der Verbindung zu einem Gerät mit der Bezeichnung `AnyDevice`.

Argumente

```
SiConnect Sensor [\NoStop]
```

`Sensor`

Datentyp: `sensor`

Der Deskriptor des externen Geräts, zu dem die Verbindung hergestellt werden soll. Das Argument ist eine persistente Variable und der Name muss dem Namen entsprechen, der als Client in der Setup-Datei *Settings.xml* angegeben wurde.

`[\NoStop]`

Datentyp: `switch`

`\NoStop` verhindert, dass das System stoppt, wenn ein Fehler in der Kommunikation mit dem Sensor entdeckt wird. Dies kann nützlich sein, wenn Roboterbewegungen vom Sensor abhängen. Wird `\NoStop` verwendet, können weiterhin Bewegungen im System erfolgen, auch wenn die Kommunikation zum Sensor verloren geht.

Wird `\NoStop` verwendet, ist es möglich, eine Fehlerbehandlung mit einer `TRAP`-Routine unter Verwendung von `IError` oder `IPers` durchzuführen.

Programmabarbeitung

Lädt die aktuelle Sensorkonfiguration und erstellt die Verbindung zum externen Gerät.

Der Sensor bleibt angeschlossen, auch wenn der Programmzeiger auf "main" gestellt wird.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `SiConnect` sind unten aufgeführt.

Beispiel 1:

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Beim Aufruf der Routine `RRI_Open` wird zunächst eine Verbindung zum Gerät mit dem Namen `AnyDevice` geöffnet. Dann beginnt eine zyklische Übertragung mit der Frequenz `SampleRate`.

Beispiel 2:

```
PERS sensor AnyDevice;
...
  SiConnect AnyDevice \NoStop;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
...
TRAP sensorChange
  IF AnyDevice.state = STATE_ERROR THEN
  ...
  ENDIF
ENDTRAP
```

Eine Verbindung zu einem Gerät mit der Bezeichnung `AnyDevice` und dem optionalen Argument `\NoStop` herzustellen verhindert, dass das System stoppt, wenn die Verbindung zu `AnyDevice` verloren geht. Fehler werden in der Routine `TRAP` behandelt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_COMM_INIT`	DEr TCP wird als Kommunikationsprotokoll verwendet, und der Verbindungsvorgang schlägt fehl.

Wird UDP als Kommunikationsprotokoll verwendet, gibt es keine Garantie, dass die Verbindung erfolgreich ist und daher ist auch keine Fehlerbehandlung bezüglich des Verbindungszeitpunktes möglich.

Der Schalter `\NoStop` ermöglicht es, mit Kommunikationsfehlern umzugehen, die nach einem erfolgreichen Verbindungsaufbau erkannt wurden. `\NoStop` bedeutet, dass RAPID weiterhin ausgeführt wird und dass eine `TRAP`-Routine verwendet

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.238 SiConnect - Sensorschnittstelle verbinden
Robot Reference Interface
Fortsetzung

werden kann, um spezifische Fehler mit `IError` oder spezifische Statuswechsel mit `IPers` zu behandeln.

> **Hinweis**
>
> `IPers` und `IError` sind keine sicheren Interrupts. Wird also ein Fehler nach einem Halt erkannt, wird `TRAP` nicht ausgeführt. Eine Möglichkeit, dieses Problem zu lösen, ist über ein `SiConnect \NoStop` im Neustart-Shelf zu verfügen, um sicherzugehen, dass die Anwendung versucht, die Verbindung zum Client wieder herzustellen.

Syntax

```
SiConnect
  [ Sensor ':=' ] < persistent (PERS) of sensor >
  [ '\' NoStop ] ';'
```

Weitere Informationen

Information über	Siehe
Schließen einer Verbindung zu einem externen System.	*SiClose - Sensorschnittstelle schließen auf Seite 731*.
Datenregistrierung für die zyklische Übertragung.	*SiSetCyclic - Sensorschnittstelle zyklisch einstellen auf Seite 739*.
Für eine zyklische Datenübertragung anmelden.	*SiGetCyclic - Sensorschnittstelle zyklisch abrufen auf Seite 733*
Deskriptor für externe Geräte.	*sensor - Externer Gerätedeskriptor auf Seite 1782*.
Kommunikationszustand eines Geräts.	*sensorstate - Kommunikationszustand des Geräts auf Seite 1784*.
Robot Reference Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.239 SiClose - Sensorschnittstelle schließen

Verwendung

`SiClose` schließt eine bestehende Verbindung zu einem externen Gerät.

Grundlegende Beispiele

Grundlegende Beispiele für die Instruktion `SiClose` ist unten aufgeführt.

Beispiel 1:

```
PERS sensor AnyDevice;
...
  SiClose AnyDevice;
```

Beenden der Verbindung zu einem Gerät mit der Bezeichnung `AnyDevice`.

Argumente

```
SiClose Sensor
```

`Sensor`

Datentyp: `sensor`

Der Deskriptor des externen Geräts, das geschlossen werden soll. Das Argument ist eine persistente Variable und der Name muss dem Namen entsprechen, der als Client in der Setup-Datei *Settings.xml* angegeben wurde.

Programmabarbeitung

Schließt eine bestehende Verbindung zum externen Gerät.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_COMM_INIT`	DEr TCP wird als Kommunikationsprotokoll verwendet, und der Verbindungsvorgang schlägt fehl.

Wird UDP als Kommunikationsprotokoll verwendet, gibt es keine Garantie, dass der Schließvorgang erfolgreich abläuft und daher ist auch keine Fehlerbehandlung möglich.

Syntax

```
SiClose
  [ Sensor ':=' ] < persistent (PERS) of sensor > ';'
```

Weitere Informationen

Information über	Siehe
Aufbau einer Verbindung zu einem externen System.	*SiConnect - Sensorschnittstelle verbinden auf Seite 728*.
Datenregistrierung für die zyklische Übertragung.	*SiSetCyclic - Sensorschnittstelle zyklisch einstellen auf Seite 739*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.239 SiClose - Sensorschnittstelle schließen

Fortsetzung

Information über	Siehe
Für eine zyklische Datenübertragung anmelden.	*SiGetCyclic - Sensorschnittstelle zyklisch abrufen auf Seite 733*.
Deskriptor für externe Geräte.	*sensor - Externer Gerätedeskriptor auf Seite 1782*.
Kommunikationszustand eines Geräts.	*sensorstate - Kommunikationszustand des Geräts auf Seite 1784*.
Robot Reference Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.240 SiGetCyclic - Sensorschnittstelle zyklisch abrufen

Verwendung

`SiGetCyclic` abonniert Daten für eine zyklische Datenübertragung von einem externen Gerät.

Grundlegende Beispiele

Ein grundlegendes Beispiel für die Instruktion `SiGetCyclic` ist unten aufgeführt.

Siehe auch *Weitere Beispiele auf Seite 734*.

Beispiel 1:

```
SiConnect AnyDevice;
! Receive data cyclic with 64 ms rate
SiGetCyclic AnyDevice, DataIn, 64;
```

Das Beispiel zeigt, wie man eine Verbindung zu einem externen Gerät herstellt und eine zyklische Datenübertragung vom Gerät `AnyDevice` einrichtet.

Argumente

```
SiGetCyclic Sensor Data Rate
```

`Sensor`

Datentyp: `sensor`

Ein Deskriptor für die externen Geräte, von denen Zyklusdaten abgerufen werden sollen. Das Argument ist eine persistente Variable und der Name muss dem Namen entsprechen, der als Client in der Setup-Datei *Settings.xml* angegeben wurde.

`Data`

Datentyp: `anytype`

Verweis auf eine Persistente mit den Daten, die vom Client erhalten werden, der im Argument `Sensor` angegeben ist. Die Variable muss als *Readable* in der Datei *Configuration.xml* definiert sein.

`Rate`

Datentyp: `num`

Übertragungsrate in Millisekunden (nur Vielfache von 4 ms werden unterstützt).

Programmabarbeitung

Die Instruktion `SiGetCyclic` abonniert Daten für eine zyklische Datenübertragung von einem externen Gerät.

Bei den Instruktionen `SiGetCyclic` und `SiSetCyclic` führt eine Übertragungsrate von 0 zum Stopp (abmelden/abbestellen) der zyklischen Übertragung der jeweiligen Daten oder des Datensatzes.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.240 SiGetCyclic - Sensorschnittstelle zyklisch abrufen

Robor Reference Interface
Fortsetzung

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `SiGetCyclic` sind unten aufgeführt.

Beispiel 1:

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Beim Aufruf der Routine `RRI_Open` wird zunächst eine Verbindung zum Gerät mit dem Namen `AnyDevice` geöffnet. Dann beginnt eine zyklische Übertragung mit der Frequenz `SampleRate`.

Syntax

```
SiGetCyclic
  [ Sensor ':=' ] < persistent (PERS) of sensor > ','
  [ Data ':=' ] < persistent (PERS) of anytype > ','
  [ Rate ':=' ] < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Aufbau einer Verbindung zu einem externen System.	*SiConnect - Sensorschnittstelle verbinden auf Seite 728*.
Schließen einer Verbindung zu einem externen System.	*SiClose - Sensorschnittstelle schließen auf Seite 731*.
Datenregistrierung für die zyklische Übertragung.	*SiSetCyclic - Sensorschnittstelle zyklisch einstellen auf Seite 739*.
Deskriptor für externe Geräte.	*sensor - Externer Gerätedeskriptor auf Seite 1782*.
Kommunikationszustand eines Geräts.	*sensorstate - Kommunikationszustand des Geräts auf Seite 1784*.
Robot Reference Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.241 SimCollision - Simulation einer Kollision

Verwendung

`SimCollision` wird für die Simulation einer Kollision verwendet. Die Instruktion kann während Tests der Fehlerbehandlung von Kollisionen wertvoll sein.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SimCollision`.

Beispiel 1:

```
SimCollision;
```

Simulieren einer Bewegungskollision für das Testen der Fehlerbehandlung.

Programmabarbeitung

Die Ausführung von `SimCollision` erzeugt eine Bewegungskollision. Die Instruktion kann während Tests der Fehlerbehandlung von Kollisionen wertvoll sein. Die Instruktion erzeugt bei der Verwendung ein Ereignisprotokoll *41910*, *Collision simulated with instruction SimCollision*.

Syntax

```
SimCollision';'
```

Weitere Informationen

Information über	Siehe
Einstellungen für die Ausführung oder Nicht-Ausführung der Fehlerbehandlung	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Controller*, Typ *General RAPID*, *CollisionErrorHandling*
Collision Detection	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.242 SingArea - Definieren der Interpolation in der Nähe singulärer Punkte

Verwendung

`SingArea` wird verwendet, um die Bewegungsart des Roboters in der Nähe von singulären Punkten zu definieren.

`SingArea` wird außerdem verwendet, um lineare und kreisförmige Interpolation für Roboter mit weniger als sechs Achsen zu definieren, und ein Roboter mit sechs Achsen kann so programmiert werden, dass er bei Sperren der Achse 4 bis 0 oder ±180 Grad betrieben werden kann.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SingArea`:

Beispiel 1:

```
SingArea \Wrist;
```

Die Orientierung des Werkzeugs kann geringfügig geändert werden, um einen singulären Punkt zu passieren (Achsen 4 und 6 gleichgerichtet).

Roboter mit weniger als sechs Achsen können evtl. eine interpolierte Werkzeugorientierung nicht erreichen. Mit `SingArea \Wrist` kann der Roboter die Bewegung ausführen, aber die Orientierung des Werkzeugs wird geringfügig geändert.

Beispiel 2:

```
SingArea \LockAxis4;
```

Ein Roboter mit sechs Achsen kann so programmiert werden, dass er bei Sperren der Achse 4 bis 0 oder ±180 Grad betrieben werden kann, um Singularitätsprobleme zu vermeiden, wenn sich Achse 5 dem Wert 0 nähert.

Die programmierte Position wird bei Sperren von Achse 4 bis 0 oder ±180 Grad erreicht. Wenn die Position nicht mit Achse 4 bei 0 oder ±180 Grad programmiert wurde, wird dies nun mit einer anderen Werkzeugorientierung erzielt.

Wenn die Ausgangsposition von Achse 4 um mehr als 2 Grad von der Sperrposition abweicht, verhält sich die erste Bewegung als wäre `SingArea` durch das Argument `\Wrist` aufgerufen worden.

Beispiel 3

```
SingArea \Off;
```

Die Werkzeugorientierung darf nicht von der programmierten Orientierung abweichen. Beim Passieren eines singulären Punkts können eine oder mehrere Achsen eine schwingende Bewegung ausführen und damit die TCP-Geschwindigkeit reduzieren.

Roboter mit weniger als sechs Achsen können evtl. eine programmierte Werkzeugorientierung nicht erreichen. In diesem Fall stoppt der Roboter.

Argumente

```
SingArea [\Wrist] | [\LockAxis4] | [\Off]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\Wrist]

Datentyp: switch

Die Werkzeugorientierung kann geringfügig geändert werden, um eine Handgelenk-Singularität zu vermeiden. Dies findet Anwendung, wenn die Achsen 4 und 6 parallel sind (Achse 5 bei 0 Grad). Wird außerdem verwendet, um lineare und kreisförmige Interpolation für Roboter mit weniger als sechs Achsen zu definieren, wenn die Werkzeugorientierung abweichen darf.

[\LockAxis4]

Datentyp: switch

Die programmierte Position wird bei Sperren von Achse 4 bis 0 oder ±180 Grad erreicht. Wenn die Position nicht mit Achse 4 bei 0 oder ±180 Grad programmiert wurde, wird dies nun mit einer anderen Werkzeugorientierung erzielt.

Wenn die Ausgangsposition von Achse 4 um mehr als 2 Grad von der Sperrposition abweicht, verhält sich die erste Bewegung als wäre SingArea durch das Argument \Wrist aufgerufen worden.

[\Off]

Datentyp: switch

Die Orientierung des Werkzeugs darf nicht abweichen. Wird verwendet, wenn keine singulären Punkte passiert werden oder wenn die Orientierung nicht geändert werden darf.

Wenn keines der Argumente angegeben ist, wird das System auf \Off gesetzt.

Programmabarbeitung

Die angegebene Interpolation gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue SingArea-Instruktion ausgeführt wird.

Wenn das Argument \Wrist angegeben ist, werden singuläre Punkte durch achsweise Interpolation vermieden. Auf diese Weise folgt das TCP dem korrekten Pfad, aber die Orientierung des Werkzeugs weicht ein wenig ab. Dies passiert auch, wenn ein singulärer Punkt nicht passiert wird.

Wenn das Argument \LockAxis4 angegeben ist, werden singuläre Punkte durch Sperren von Achse 4 auf 0 oder ±180 Grad vermieden. Wenn die Position nicht mit Achse 4 bei 0 oder ±180 Grad programmiert wurde, folgt der TCP der korrekten Bahn, aber die Werkzeugorientierung weicht ab. Bei Pfaden mit großen Umorientierungen kann der TCP von der programmierten linearen Bahn abweichen.

Die Bewegung wird nur bei der Abarbeitung einer linearen oder kreisförmigen Interpolation beeinflusst.

Standardmäßig verwendet die Programmabarbeitung für Roboter mit sechs Achsen automatisch das Argument Off. Roboter mit weniger als sechs Achsen können standardmäßig das Argument Off oder das Argument /Wrist verwenden. Dies wird automatisch in der Ereignisroutine SYS_RESET bei folgenden Ereignissen festgelegt:

Der Standardwert wird automatisch eingestellt

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.242 SingArea - Definieren der Interpolation in der Nähe singulärer Punkte

RobotWare - OS

Fortsetzung

- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Syntax

```
SingArea
  [ '\' Wrist ] | ['\' LockAxis4 ] | ['\' Off ] ';'
```

Weitere Informationen

Information über	Siehe
Singularität	*Technisches Referenzhandbuch - RAPID Overview*
Interpolation	*Technisches Referenzhandbuch - RAPID Overview*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.243 SiSetCyclic - Sensorschnittstelle zyklisch einstellen

Verwendung

`SiSetCyclic` registriert Daten für eine zyklische Datenübertragung an ein externes Gerät.

Grundlegende Beispiele

Ein grundlegendes Beispiel für die Instruktion `SiSetCyclic` ist unten aufgeführt.

Siehe auch *Weitere Beispiele auf Seite 740*.

Beispiel 1:

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
...
  SiConnect AnyDevice;
  SiSetCyclic AnyDevice, DataOut, 40;
```

Stellen Sie eine Verbindung zu einem Gerät mit der Bezeichnung `AnyDevice` her. Registrieren Sie Daten für eine zyklische Datenübertragung an ein externes Gerät `AnyDevice` alle 40 ms.

Argumente

```
SiSetCyclic Sensor Data Rate
```

`Sensor`

Datentyp: `sensor`

Ein Deskriptor für das externe Gerät, an das Daten gesendet werden sollen.

`Data`

Datentyp: `anytype`

Verweis auf eine Persistente eines beliebigen unterstützten komplexen oder einfachen Typs mit den Daten, die an den Client gesendet werden, der im Argument `Sensor` angegeben ist. Die Variable muss als *Writable* in der Datei *Configuration.xml* definiert sein.

`Rate`

Datentyp: `num`

Übertragungsrate in Millisekunden (nur Vielfache von 4 ms werden unterstützt).

Programmabarbeitung

Die Instruktion `SiSetCyclic` registriert Daten für eine zyklische Datenübertragung an ein externes Gerät.

Bei den Instruktionen `SiGetCyclic` und `SiSetCyclic` führt eine Übertragungsrate von 0 zum Stopp (abmelden/abbestellen) der zyklischen Übertragung der jeweiligen Daten oder des Datensatzes.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.243 SiSetCyclic - Sensorschnittstelle zyklisch einstellen

Robor Reference Interface
Fortsetzung

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `SiSetCyclic` sind unten aufgeführt.

Beispiel 1:

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Beim Aufruf der Routine `RRI_Open` wird zunächst eine Verbindung zum Gerät mit dem Namen `AnyDevice` geöffnet. Dann beginnt eine zyklische Übertragung mit der Frequenz `SampleRate`.

Syntax

```
SiSetCyclic
  [ Sensor ':=' ] < persistent (PERS) of sensor > ','
  [ Data ':=' ] < persistent (PERS) of anytype >
  [ Rate ':=' ] < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Aufbau einer Verbindung zu einem externen System.	*SiConnect - Sensorschnittstelle verbinden auf Seite 728*.
Schließen einer Verbindung zu einem externen System.	*SiClose - Sensorschnittstelle schließen auf Seite 731*.
Für eine zyklische Datenübertragung anmelden.	*SiGetCyclic - Sensorschnittstelle zyklisch abrufen auf Seite 733*.
Deskriptor für externe Geräte.	*sensor - Externer Gerätedeskriptor auf Seite 1782*.
Kommunikationszustand eines Geräts.	*sensorstate - Kommunikationszustand des Geräts auf Seite 1784*.
Robot Reference Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.244 SkipWarn - Überspringen der neuesten Warnmeldung

Verwendung

`SkipWarn`(*Skip Warning*) wird verwendet, um die zuletzt generierte Warnmeldung zu überspringen, die im Ereignisprotokoll während der Abarbeitung im kontinuierlichen oder zyklischen Modus gespeichert werden soll (keine übersprungenen Warnungen bei Vorwärts- oder Rückwärtsschritt).

Mit `SkipWarn` lässt sich in RAPID wiederholt eine Fehlerbehandlung durchführen, ohne das Ereignisprotokoll nur mit Warnungen zu füllen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SkipWarn`:

Beispiel 1:

```
  %"notexistingproc"%;
  nextinstruction;
ERROR
  IF ERRNO = ERR_REFUNKPRC THEN
    SkipWarn;
    TRYNEXT;
  ENDIF
ENDPROC
```

Das Programm arbeitet die `nextinstruction` ab und im Ereignisprotokoll wird keine Warnmeldung gespeichert.

Syntax

```
SkipWarn ';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview* *Technisches Referenzhandbuch - RAPID Overview*
Fehlernummer	*errnum – Fehlernummer auf Seite 1694*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.245 SocketAccept - Akzeptieren einer eingehenden Verbindung

Verwendung

`SocketAccept` wird verwendet, um eingehende Verbindungsanforderungen zu akzeptieren. `SocketAccept` kann nur für Serveranwendungen verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketAccept`:

Siehe auch *Weitere Beispiele auf Seite 743*.

Beispiel 1:

```
VAR socketdev server_socket;
VAR socketdev client_socket;
...
SocketCreate server_socket;
SocketBind server_socket,"192.168.0.1", 1025;
SocketListen server_socket;
SocketAccept server_socket, client_socket;
```

Ein Server-Socket wird erstellt und an Port `1025` im Steuerungsnetzwerk mit der Adresse `192.168.0.1` gebunden. Nach der Abarbeitung von `SocketListen` beginnt der Server mit der Überwachung eingehender Verbindungen an diesem Port und dieser Adresse. `SocketAccept` wartet auf eingehende Verbindungen, akzeptiert die Verbindungsanfrage und gibt ein Client-Socket für die hergestellte Verbindung zurück.

Argumente

```
SocketAccept Socket ClientSocket [\ClientAddress] [ \Time ]
```

`Socket`

Datentyp: `socketdev`

Der Server-Socket, der auf eingehende Verbindungen wartet. Der Socket muss bereits erstellt, gebunden und für die Überprüfung bereit sein.

`ClientSocket`

Datentyp: `socketdev`

Der bereitgestellte neue Client-Socket, der mit der akzeptierten eingehenden Verbindungsanfrage aktualisiert wird.

`[\ClientAddress]`

Datentyp: `string`

Die Variable, die mit der IP-Adresse der akzeptierten eingehenden Verbindungsanfrage aktualisiert wird.

`[\Time]`

Datentyp: `num`

Die maximale Zeitdauer [s], für die die Programmabarbeitung auf eingehende Verbindungen wartet. Wenn diese Zeit ohne eingehende Verbindung abläuft, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ERR_SOCK_TIMEOUT aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Wenn der Parameter \Time nicht verwendet wird, beträgt die Wartezeit 60 s. Um für eine unendliche Dauer zu warten, verwenden Sie die vordefinierte Konstante WAIT_MAX.

Programmabarbeitung

Der Server-Socket wartet auf eingehende Verbindungsanfragen. Wenn eine eingehende Verbindungsanfrage akzeptiert wird, ist die Instruktion bereit und der zurückgegebene Client-Socket wird standardmäßig verbunden und kann in SocketSend- und SocketReceive-Instruktionen verwendet werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SOCK_CLOSED	Der Socket ist geschlossen (wurde geschlossen oder nicht erstellt). Verwenden Sie SocketCreate zum Erstellen eines neuen Sockets.
ERR_SOCK_TIMEOUT	Die Verbindung wurde nicht innerhalb der Timeout-Zeit hergestellt.
ERR_SOCK_UNSPEC	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.

Weitere Beispiele

Weitere Beispiele für die Instruktion SocketAccept sind unten aufgeführt.

Beispiel 1:

```
VAR socketdev server_socket;
VAR socketdev client_socket;
VAR string receive_string;
VAR string client_ip;
...
SocketCreate server_socket;
SocketBind server_socket, "192.168.0.1", 1025;
SocketListen server_socket;
WHILE TRUE DO
  SocketAccept server_socket, client_socket
       \ClientAddress:=client_ip;
  SocketReceive client_socket \Str := receive_string;
  SocketSend client_socket \Str := "Hello client with ip-address
       " +client_ip;
  ! Wait for client acknowledge
  ...
  SocketClose client_socket;
ENDWHILE
ERROR
  RETRY;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.245 SocketAccept - Akzeptieren einer eingehenden Verbindung

Socket Messaging

Fortsetzung

```
UNDO
  SocketClose server_socket;
  SocketClose client_socket;
```

Ein Server-Socket wird erstellt und an Port `1025` im Steuerungsnetzwerk mit der Adresse `192.168.0.1` gebunden. Nach der Abarbeitung von `SocketListen` beginnt der Server mit der Überwachung eingehender Verbindungen an diesem Port und dieser Adresse. `SocketAccept` akzeptiert die eingehende Verbindung von einem Client und speichert die Client-Adresse in der Zeichenfolge `client_ip`. Anschließend empfängt der Server eine Zeichenfolgenmeldung vom Client und speichert die Meldung in `receive_string`. Daraufhin antwortet der Server mit der Meldung `" Hello client with ip-address xxx.xxx.x.x"` und schließt die Client-Verbindung.

Anschließend ist der Server für eine Verbindung von demselben oder einem anderen Client in der `WHILE`-Schleife bereit. Wenn der Programmzeiger im Programm auf Main gesetzt wird, werden alle offenen Sockets geschlossen (`SocketClose` kann immer ausgeführt werden, auch wenn der Socket nicht erstellt wurde).

Syntax

```
SocketAccept
  [Socket ':='] <variable (VAR) of socketdev>','
  [ClientSocket ':='] <variable (VAR) of socketdev>
  ['\' ClientAddress ':=' <variable (VAR) of string>]
  ['\' Time ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *Socket Messaging*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.246 SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port

Verwendung

`SocketBind` wird verwendet, um einen Socket an die angegebene Serverportnummer und IP-Adresse zu binden. `SocketBind` kann nur für Serveranwendungen verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketBind`:

Beispiel 1:

```
VAR socketdev server_socket;

SocketCreate server_socket;
SocketBind server_socket, "192.168.0.1", 1025;
```

Ein Server-Socket wird erstellt und an Port `1025` im Steuerungsnetzwerk mit der Adresse `192.168.0.1` gebunden. Der Server-Socket kann jetzt in einer `SocketListen`-Instruktion verwendet werden, um diesen Port und diese Adresse auf eingehende Verbindungen zu überprüfen.

Argumente

```
SocketBind Socket LocalAddress LocalPort
```

`Socket`

Datentyp: `socketdev`

Der zu bindende Server-Socket. Der Socket muss erstellt, darf aber noch nicht gebunden sein.

`LocalAddress`

Datentyp: `string`

Die Netzwerkadresse des Servers, an die der Socket gebunden werden soll. Die einzigen gültigen Adressen sind beliebige öffentliche WAN-Adressen oder die Serviceportadresse 192.168.125.1 der Steuerung.

`LocalPort`

Datentyp: `num`

Die Portnummer des Servers, an die der Socket gebunden werden soll. Normalerweise stehen die Ports 1025-4999 zur freien Verfügung.

Programmabarbeitung

Der Server-Socket wird an den angegebenen Serverport und die angegebene IP-Adresse gebunden.

Wenn der angegebene Port bereits belegt ist, wird eine Fehlermeldung generiert.

Verwenden Sie zu Beginn des Programms die Instruktionen `SocketBind` und `SocketListen`, um einem Socket eine lokale Adresse zuzuordnen und dann am angegebenen Port eingehende Verbindungen zu überwachen. Dies sollte für jeden verwendeten Socket und Port nur einmal ausgeführt werden (TCP/IP).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.246 SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port

Socket Messaging
Fortsetzung

Verwenden Sie die Instruktion `SocketBind`, wenn Daten mit `SocketReceiveFrom` empfangen werden (UDP/IP).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_SOCK_ADDR_INVALID`	Die angegebene Adresse ist ungültig.
`ERR_SOCK_ADDR_INUSE`	Die Adresse und der Port werden bereits verwendet und können nicht erneut verwendet werden. Verwenden Sie eine andere Portnummer.
`ERR_SOCK_IS_BOUND`	Der Socket ist bereits an eine Adresse gebunden und kann nicht erneut gebunden werden.
`ERR_SOCK_CLOSED`	Der Socket ist geschlossen (wurde geschlossen oder nicht erstellt). Verwenden Sie `SocketCreate`, um einen neuen Socket zu erstellen.
`ERR_SOCK_UNSPEC`	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.

Syntax

```
SocketBind
  [Socket ':='] <variable (VAR) of socketdev>','
  [LocalAddress ':='] <expression (IN) of string>','
  [LocalPort ':='] <expression (IN) of num>';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.246 SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port

Socket Messaging

Fortsetzung

Information über	Siehe
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Daten vom Remote-Computer empfangen	*SocketReceiveFrom - Daten vom Remote-Computer empfangen auf Seite 764*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.247 SocketClose - Schließen eines Sockets

Verwendung

`SocketClose` wird verwendet, wenn eine Socketverbindung nicht länger verwendet wird.

Nachdem ein Socket geschlossen wurde, kann er in keinem Socket-Aufruf mehr verwendet werden, außer in `SocketCreate`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketClose`:

Beispiel 1:

```
SocketClose socket1;
```

Die Buchse ist geschlossen und kann nicht mehr verwendet werden.

Argumente

```
SocketClose Socket
```

`Socket`

Datentyp: `socketdev`

Der zu schließende Socket.

Programmabarbeitung

Der Socket wird geschlossen und die ihr zugewiesenen Ressourcen werden freigegeben.

Buchsen können jederzeit geschlossen werden. Buchsen können nach dem Schließen nicht verwendet werden. Nach einem Aufruf von `SocketCreate` kann sie für eine neue Verbindung wiederverwendet werden.

Einschränkungen

Wenn direkt nach dem Senden der Daten mit `SocketSend` die Socketverbindung geschlossen wird, kann dies zum Verlust der gesendeten Daten führen. Der Grund dafür ist, dass `TCP/IP`-Sockets die Daten erneut senden, wenn Verbindungsprobleme auftreten.

Um den Verlust von Daten zu vermeiden, führen Sie vor `SocketClose` Folgendes aus:

- Handshake des Herunterfahrens oder
- `WaitTime` 2

Vermeiden Sie schnelle Schleifen mit `SocketCreate ... SocketClose`, da der Socket nicht vor einem bestimmten Zeitpunkt geschlossen wird (TCP/IP-Funktionalität).

Syntax

```
SocketClose
  [Socket ':='] <variable (VAR) of socketdev>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.247 SocketClose - Schließen eines Sockets

Socket Messaging

Fortsetzung

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *Socket Messaging*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zu einem Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*t
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Senden von Daten an Remote-Computer	*SocketSendTo - Daten an Remote-Computer senden auf Seite 774*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Daten vom Remote-Computer empfangen	*SocketReceiveFrom - Daten vom Remote-Computer empfangen auf Seite 764*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.248 SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer

Verwendung

`SocketConnect` wird verwendet, um einen Socket mit einem Remote-Computer in einer Client-Anwendung zu verbinden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketConnect`:

Siehe auch *Weitere Beispiele auf Seite 752*.

Beispiel 1:

```
SocketConnect socket1, "192.168.0.1", 1025;
```

Es wird versucht, über die IP-Adresse `192.168.0.1` und den Port `1025` eine Verbindung zu einem Remote-Computer herzustellen.

Argumente

```
SocketConnect Socket Address Port [\Time]
```

`Socket`

Datentyp: `socketdev`

Der Client-Socket, zu der eine Verbindung hergestellt werden soll. Der Socket muss erstellt sein, es darf jedoch noch keine Verbindung dazu hergestellt worden sein.

`Address`

Datentyp: `string`

Die Adresse des Remote-Computers. Der Remote-Computer muss durch eine IP-Adresse angegeben werden. Der Name des Remote-Computers kann nicht verwendet werden.

`Port`

Datentyp: `num`

Der Port am Remote-Computer. Normalerweise stehen die Ports 1025-4999 zur freien Verfügung. Portnummern unter 1025 können bereits belegt sein.

`[ \Time ]`

Datentyp: `num`

Die maximale Zeitdauer [s], für die die Programmabarbeitung wartet, ob die Verbindung akzeptiert oder abgelehnt wird. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_SOCK_TIMEOUT` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Wenn der Parameter `\Time` nicht verwendet wird, beträgt die Wartezeit 60 s. Um für eine unendliche Dauer zu warten, verwenden Sie die vordefinierte Konstante `WAIT_MAX`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.248 SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer
Socket Messaging
Fortsetzung

Programmabarbeitung

Der Socket versucht, über die angegebene IP-Adresse und den angegebenen Port eine Verbindung zum Remote-Computer herzustellen. Die Programmabarbeitung wartet, bis die Verbindung hergestellt wurde, fehlschlägt oder ein Timeout eintritt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SOCK_ADDR_INVALID	Die angegebene Adresse ist ungültig
ERR_SOCK_CLOSED	Der Socket ist geschlossen (wurde geschlossen oder nicht erstellt). Verwenden Sie SocketCreate, um einen neuen Socket zu erstellen.
ERR_SOCK_IS_CONN	Der Socket ist verbunden.
ERR_SOCK_NET_UNREACH	Das Netzwerk ist nicht erreichbar oder die Verbindung geht nach dem Öffnen eines Sockets verloren.
ERR_SOCK_TIMEOUT	Die Verbindung wurde nicht innerhalb der Timeout-Zeit hergestellt.
ERR_SOCK_UNSPEC	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.

Weitere Beispiele

Weitere Beispiele für die Instruktion SocketConnect sind unten aufgeführt.

Beispiel 1:

```
VAR num retry_no := 0;
VAR socketdev my_socket;
...
SocketCreate my_socket;
SocketConnect my_socket, "192.168.0.1", 1025;
...
ERROR
  IF ERRNO = ERR_SOCK_TIMEOUT THEN
    IF retry_no < 5 THEN
      WaitTime 1;
      retry_no := retry_no + 1;
      RETRY;
    ELSE
      RAISE;
    ENDIF
  ENDIF
```

Ein Socket wird erstellt und versucht, eine Verbindung zum Remote-Computer herzustellen. Wenn die Verbindung nicht innerhalb der standardmäßigen Timeout-Zeit von 60 Sekunden hergestellt werden kann, versucht die Fehlerbehandlung, eine Verbindung herzustellen. Der Versuch, eine Verbindung herzustellen, wird viermal wiederholt, bevor der Benutzer benachrichtigt wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.248 SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer

Socket Messaging

Fortsetzung

Syntax

```
SocketConnect
  [Socket ':='] <variable (VAR) of socketdev>','
  [Address ':='] <expression (IN) of string>','
  [Port ':='] <expression (IN) of num>
  ['\' Time ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Beschrieben in:
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.249 SocketCreate - Erstellen eines neuen Sockets

Verwendung

`SocketCreate` wird verwendet, um für die verbindungsbasierte Kommunikation oder verbindungslose Kommunikation einen neuen Socket zu erstellen.

Es wird sowohl die Socketkommunikation über TCP/IP mit garantierter Übertragung als auch über UDP/IP unterstützt. Sowohl der Server als auch die Client-Anwendung können entwickelt werden. Die Übertragung von UDP/IP wird unterstützt.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SocketCreate`:

Beispiel 1:

```
VAR socketdev socket1;
...
SocketCreate socket1;
```

Ein neues Socketgerät mit TCP/IP wird erstellt und der Variable `socket1` zugewiesen.

Beispiel 2:

```
VAR socketdev udp_sock1;
...
SocketCreate udp_sock1 \UDP;
```

Ein neues Socketgerät mit UDP/IP wird erstellt und der Variable `udp_sock1` zugewiesen.

Argumente

```
SocketCreate Socket [\UDP]
```

`Socket`

Datentyp: `socketdev`

Die Variable zur Speicherung systeminterner Socketdaten.

`[ \UDP ]`

Datentyp: `switch`

Gibt an, dass der Socket vom Typ UDP/IP sein soll.

Programmabarbeitung

Die Instruktion erstellt ein neues Socket-Gerät.

Der Socket darf noch nicht verwendet werden. Verwendet wird der Socket zwischen der Abarbeitung der Instruktionen `SocketCreate` und `SocketClose`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SOCK_UNSPEC	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.

Einschränkungen

Sie können eine beliebige Anzahl an Sockets deklarieren, es können jedoch nur 32 Sockets gleichzeitig verwendet werden.

Vermeiden Sie schnelle Schleifen mit `SocketCreate ... SocketClose`, da der Socket nicht vor einem bestimmten Zeitpunkt geschlossen wird (bei Verwendung der TCP/IP-Funktionalität).

Syntax

```
SocketCreate
  [Socket ':='] <variable (VAR) of socketdev>
  ['\' UDP]';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *Socket Messaging*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Senden von Daten an Remote-Computer	*SocketSendTo - Daten an Remote-Computer senden auf Seite 774*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.249 SocketCreate - Erstellen eines neuen Sockets

Socket Messaging

Fortsetzung

Information über	Siehe
Daten vom Remote-Computer empfangen	*SocketReceiveFrom - Daten vom Remote-Computer empfangen auf Seite 764*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.250 SocketListen - Prüfen auf eingehende Verbindungen

Verwendung

`SocketListen` wird verwendet, um mit der Überprüfung auf eingehende Verbindungen zu beginnen (d.h., als Server zu fungieren). `SocketListen` kann nur für Server-Anwendungen verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketListen`:

Beispiel 1:

```
VAR socketdev server_socket;
VAR socketdev client_socket;
...
SocketCreate server_socket;
SocketBind server_socket, "192.168.0.1", 1025;
SocketListen server_socket;
WHILE listening DO;
  ! Waiting for a connection request
  SocketAccept server_socket, client_socket;
```

Ein Server-Socket wird erstellt und an Port `1025` im Steuerungsnetzwerk mit der Adresse `192.168.0.1` gebunden. Nach der Abarbeitung von `SocketListen` überprüft der Server-Socket diesen Port und diese Adresse auf eingehende Verbindungen.

Argumente

```
SocketListen Socket
```

`Socket`

Datentyp: `socketdev`

Der Server-Socket, der auf eingehende Verbindungen überprüfen soll. Das Socket muss bereits erstellt und gebunden worden sein.

Programmabarbeitung

Der Server-Socket beginnt mit der Überprüfung auf eingehende Verbindungen. Wenn die Instruktion bereit ist, kann der Socket eine eingehende Verbindung akzeptieren.

Verwenden Sie zu Beginn des Programms die Instruktionen `SocketBind` und `SocketListen`, um einem Socket eine lokale Adresse zuzuordnen und dann am angegebenen Port eingehende Verbindungen zu überwachen. Dies sollte für jeden verwendeten Socket und Port nur einmal ausgeführt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.250 SocketListen - Prüfen auf eingehende Verbindungen

Socket Messaging

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SOCK_CLOSED	Der Socket ist geschlossen (wurde geschlossen oder nicht erstellt). Verwenden Sie SocketCreate, um einen neuen Socket zu erstellen.
ERR_SOCK_IS_CONN	Der Socket ist verbunden.
ERR_SOCK_NOT_BOUND	Der Socket wurde nicht an eine Adresse gebunden.
ERR_SOCK_UNSPEC	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.

Syntax

```
SocketListen
  [Socket ':='] <variable (VAR) of socketdev>';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.251 SocketReceive - Empfangen von Daten vom Remote-Computer

Verwendung

`SocketReceive` wird verwendet, um Daten von einem Remote-Computer zu empfangen. `SocketReceive` kann sowohl für Client- als auch für Serveranwendungen verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketReceive`:

Siehe auch *Weitere Beispiele auf Seite 761*.

Beispiel 1:

```
VAR string str_data;
...
SocketReceive socket1 \Str := str_data;
```

Daten von einem Remote-Computer empfangen und in der Zeichenfolgenvariable `str_data` speichern.

Argumente

```
SocketReceive Socket [ \Str ] | [ \RawData ] | [ \Data ]
     [\ReadNoOfBytes] [\NoRecBytes] [\Time]
```

`Socket`

Datentyp: `socketdev`

In Client-Anwendungen, in denen die Daten vom Socket empfangen werden, muss der Socket, der die Daten empfängt, bereits erstellt und verbunden sein.

In Server-Anwendungen muss der Socket, der die Daten empfängt, bereits akzeptiert sein.

`[ \Str ]`

Datentyp: `string`

Die Variable, in der die empfangenen `string`-Daten gespeichert werden sollen. Es können maximal 80 Zeichen behandelt werden.

`[ \RawData ]`

Datentyp: `rawbytes`

Die Variable, in der die empfangenen rawbytes-Daten gespeichert werden sollen. Es können maximal 1024 `rawbytes` behandelt werden.

`[ \Data ]`

Datentyp: `array of byte`

Die Variable, in der die empfangenen byte-Daten gespeichert werden sollen. Es können maximal 1024 `byte` behandelt werden.

Es kann jeweils nur einer der optionalen Parameter `\Str`, `\RawData` und `\Data` verwendet werden.

`[ \ReadNoOfBytes ]`

Read number of Bytes

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.251 SocketReceive - Empfangen von Daten vom Remote-Computer
Socket Messaging
Fortsetzung

Datentyp: num

Die Anzahl der zu lesenden Bytes. Die Mindestanzahl der zu lesenden Bytes ist 1 und die maximale Anzahl ist der Wert für die Größe des verwendeten Datentyps, d. h. 80 Bytes, wenn eine Variable vom Datentyp string verwendet wird.

Bei der Kommunikation mit einem Client, der immer eine feste Anzahl von Bytes sendet, kann mit diesem optionalen Parameter angegeben werden, dass für jede SocketReceive-Instruktion dieselbe Anzahl von Bytes gelesen werden soll.

Wenn der Sender RawData sendet, muss der Empfänger angeben, dass für jedes gesendete rawbytes 4 Bytes gelesen werden müssen.

[\NoRecBytes]

Number Received Bytes

Datentyp: num

Variable zum Speichern der Anzahl der erforderlichen Bytes aus dem angegebenen socketdev.

Dasselbe Ergebnis kann auch erreicht werden mit

- der Funktion StrLen für die Variable im Argument \Str
- der Funktion RawBytesLen für die Variable im Argument \RawData

[\Time]

Datentyp: num

Die maximale Zeitdauer [s], für die die Programmabarbeitung auf die zu empfangenden Daten wartet. Wenn diese Zeit abläuft, bevor die Daten übertragen wurden, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode ERR_SOCK_TIMEOUT aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Wenn der Parameter \Time nicht verwendet wird, beträgt die Wartezeit 60 s. Um für eine unendliche Dauer zu warten, verwenden Sie die vordefinierte Konstante WAIT_MAX.

Programmabarbeitung

Die Abarbeitung von SocketReceive wartet, bis die Daten verfügbar sind, oder sie schlägt ohne Timeout-Fehler fehl.

Die Anzahl der zu lesenden Bytes wird von dem in der Instruktion verwendeten Datentyp angegeben. Wenn zum Empfangen von Daten der Datentyp string verwendet wird, werden 80 Byte empfangen, wenn 80 Byte gelesen werden können. Bei Verwendung des optionalen Arguments ReadNoOfBytes kann der Benutzer die Anzahl der Bytes angeben, die für jedes SocketReceive empfangen werden sollen.

Die über das Kabel übertragenen Daten sind immer byte-Daten. Eine Meldung kann aus maximal 1024 Byte bestehen. Der Meldung wird standardmäßig kein Header hinzugefügt. Die Verwendung von Headern ist für die eigentliche Anwendung reserviert.

Parameter	Eingangsdaten	Kabeldaten	Ausgangsdaten
\Str	1 char	1 byte (8 bits)	1 char

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Parameter	Eingangsdaten	Kabeldaten	Ausgangsdaten
\RawData	1 rawbytes	1 byte (8 bits)	1 rawbytes
\Data	1 Byte	1 byte (8 bits)	1 Byte

In SocketSend und SocketReceive können verschiedene Datentypen gemeinsam verwendet werden (string, rawbytes oder array of byte).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SOCK_CLOSED	Der Socket ist geschlossen. Verbindung ist unterbrochen.
ERR_SOCK_NET_UNREACH	Das Netzwerk ist nicht erreichbar oder die Verbindung geht nach dem Öffnen eines Sockets verloren.
ERR_SOCK_NOT_CONN	Der Socket ist nicht verbunden.
ERR_SOCK_TIMEOUT	Die Daten wurden nicht innerhalb der Timeout-Zeit empfangen.

Weitere Beispiele

Weitere Beispiele für die Instruktion SocketReceive sind unten aufgeführt.

Beispiel 1:

```
VAR socketdev server_socket;
VAR socketdev client_socket;
VAR string client_ip;

PROC server_messaging()
  VAR string receive_string;
  ...
  ! Create, bind, listen and accept of sockets in error handlers
  SocketReceive client_socket \Str := receive_string;
  SocketSend client_socket \Str := "Hello client with
       ip-address"+client_ip;
  ! Wait for acknowlegde from client
  ...
  SocketClose server_socket;
  SocketClose client_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    server_recover;
    RETRY;
  ELSE
  ! No error recovery handling
  ENDIF
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.251 SocketReceive - Empfangen von Daten vom Remote-Computer

Socket Messaging

Fortsetzung

```
PROC server_recover()
  SocketClose server_socket;
  SocketClose client_socket;
  SocketCreate server_socket;
  SocketBind server_socket, "192.168.0.1", 1025;
  SocketListen server_socket;
  SocketAccept server_socket,
       client_socket\ClientAddress:=client_ip;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Hierbei handelt es sich um ein Serverprogramm zum Erstellen, Binden, Prüfen und Akzeptieren von Sockets in Fehlerbehandlungen. Auf diese Weise kann das Programm einen Neustart nach einem Netzausfall behandeln.

In der Prozedur `server_recover` wird ein Server-Socket erstellt und an Port `1025` im Steuerungsnetzwerk mit der Adresse `192.168.0.1` gebunden. Nach der Abarbeitung von `SocketListen` beginnt der Server mit der Überwachung eingehender Verbindungen an diesem Port und dieser Adresse. `SocketAccept` akzeptiert die eingehende Verbindung von einem Client und speichert die Client-Adresse in der Zeichenfolge `client_ip`.

In der Kommunikationsprozedur `server_messaging` empfängt der Server eine Zeichenfolgenmeldung vom Client und speichert die Meldung in `receive_string`. Anschließend antwortet der Server mit der Meldung `"Hello client with ip-address xxx.xxx.x.x"`.

Einschränkungen

Die Socket-Kommunikation verfügt über keinen integrierten Synchronisierungsmechanismus, um den Empfang von Meldungen zu vermeiden, die aus mehreren gesendeten Meldungen bestehen. Es fällt in den Aufgabenbereich des Programmierers, die Synchronisierung durch Bestätigungsmeldungen zu regeln (eine `SocketSend - SocketReceive`-Sequenz im Client- oder Serverprogramm muss abgeschlossen sein, bevor die nächste `SocketSend - SocketReceive`-Sequenz begonnen werden kann).

Nach einem Neustart nach einem Netzausfall sind alle Sockets geschlossen. Dieses Problem kann von der Fehlerbehandlung behoben werden. Siehe obiges Beispiel.

Vermeiden Sie schnelle Schleifen mit `SocketCreate ... SocketClose`, da der Socket nicht vor einem bestimmten Zeitpunkt geschlossen wird (TCP/IP-Funktionalität).

Die maximale Datenmenge, die innerhalb eines Aufrufs empfangen werden kann, ist auf 1024 Byte beschränkt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.251 SocketReceive - Empfangen von Daten vom Remote-Computer

Socket Messaging

Fortsetzung

Syntax

```
SocketReceive
  [Socket ':='] <variable (VAR) of socketdev>
  ['\' Str ':=' <variable (VAR) of string>]
  |['\' RawData ':=' <variable (VAR) of rawbytes>]
  |['\' Data ':=' <array {*} (VAR) of byte>]
  ['\' ReadNoOfBytes ':=' <expression (IN) of num>]
  ['\' NoRecBytes ':=' <variable (VAR) of num>]
  ['\' Time ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Überprüfung eines Sockets auf vorhandene Daten hin.	*SocketPeek – Überprüfung eines Sockets auf vorhandene Daten hin auf Seite 1499*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.252 SocketReceiveFrom - Daten vom Remote-Computer empfangen

Verwendung

`SocketReceiveFrom` wird verwendet, um Daten von einem Remote-Computer zu empfangen. `SocketReceiveFrom` kann sowohl für Client- als auch für Serveranwendungen verwendet werden.

`SocketReceiveFrom` wird für verbindungslose Kommunikation mit UDP/IP verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketReceiveFrom`:

Siehe auch *Weitere Beispiele auf Seite 761*.

Beispiel 1:

```
VAR string str_data;
VAR string RemoteAddress;
VAR num RemotePort;
...
SocketCreate \UDP;
SocketBind myUDPsock, "192.168.9.100", 4044;
SocketReceiveFrom socket1 \Str := str_data, RemoteAddress,
     RemotePort;
```

Daten von einem Remote-Computer empfangen und in der Zeichenfolgenvariable `str_data` speichern. Die Adresse des Remote-Computers wird in der Zeichenfolgenvariable `RemoteAddress` gespeichert, und die Portnummer wird in der numerischen Variable `RemotePort` gespeichert.

Argumente

```
SocketReceiveFrom Socket [ \Str ] | [ \RawData ] | [ \Data ]
     [\ReadNoOfBytes] [\NoRecBytes] RemoteAddress RemotePort
     [\Time]
```

`Socket`

Datentyp: `socketdev`

Ein Socketgerät, das ein verbundenen Socket identifiziert.

`[ \Str ]`

Datentyp: `string`

Die Variable, in der die empfangenen `string`-Daten gespeichert werden sollen. Es können maximal 80 Zeichen behandelt werden.

`[ \RawData ]`

Datentyp: `rawbytes`

Die Variable, in der die empfangenen rawbytes-Daten gespeichert werden sollen. Es können maximal 1024 `rawbytes` behandelt werden.

`[ \Data ]`

Datentyp: `array of byte`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Variable, in der die empfangenen byte-Daten gespeichert werden sollen. Es können maximal 1024 `byte` behandelt werden.

Es kann jeweils nur einer der optionalen Parameter `\Str`, `\RawData` und `\Data` verwendet werden.

`[ \ReadNoOfBytes ]`

Read number of Bytes

Datentyp: `num`

Die Anzahl der zu lesenden Bytes. Die Mindestanzahl der zu lesenden Bytes ist 1 und die maximale Anzahl ist der Wert für die Größe des verwendeten Datentyps, d. h. 80 Bytes, wenn eine Variable vom Datentyp `string` verwendet wird.

Bei der Kommunikation mit einem Client, der immer eine feste Anzahl von Bytes sendet, kann mit diesem optionalen Parameter angegeben werden, dass für jede `SocketReceive`-Instruktion dieselbe Anzahl von Bytes gelesen werden soll.

Wenn der Sender RawData sendet, muss der Empfänger angeben, dass für jedes gesendete `rawbytes` 4 Bytes gelesen werden müssen.

`[ \NoRecBytes ]`

Number Received Bytes

Datentyp: `num`

Variable zum Speichern der Anzahl der erforderlichen Bytes aus dem angegebenen `socketdev`.

Dasselbe Ergebnis kann auch erreicht werden mit

- der Funktion `StrLen` für die Variable im Argument `\Str`
- der Funktion `RawBytesLen` für die Variable im Argument `\RawData`

`RemoteAddress`

Datentyp: `string`

Eine Zeichenfolgenvariable, die die Quelladresse des Remote-Computers beinhaltet.

`RemotePort`

Datentyp: `num`

Eine numerische Variable, die den vom Remote-Computer verwendeten Port beinhaltet, wenn das Datenpaket verschickt wird.

`[ \Time ]`

Datentyp: `num`

Die maximale Zeitdauer [s], für die die Programmabarbeitung auf die zu empfangenden Daten wartet. Wenn diese Zeit abläuft, bevor die Daten übertragen wurden, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_SOCK_TIMEOUT` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Wenn der Parameter `\Time` nicht verwendet wird, beträgt die Wartezeit 60 s. Um für eine unendliche Dauer zu warten, verwenden Sie die vordefinierte Konstante `WAIT_MAX`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.252 SocketReceiveFrom - Daten vom Remote-Computer empfangen

Socket Messaging
Fortsetzung

Programmabarbeitung

Die Abarbeitung von `SocketReceiveFrom` erhält ein Datenpaket und speichert die Quelladresse und den Quellport. Die Abarbeitung wartet, bis die Daten verfügbar sind, oder sie schlägt durch einen Zeitüberschreitungsfehler fehl.

Die Anzahl der zu lesenden Bytes wird vom in der Instruktion verwendeten Datentyp angegeben. Wenn zum Empfangen von Daten der Datentyp `string` verwendet wird, werden 80 Byte empfangen, wenn 80 Byte gelesen werden können.

Die über das Kabel übertragenen Daten sind immer byte-Daten. Eine Meldung kann aus maximal 1024 Byte bestehen. Der Meldung wird standardmäßig kein Header hinzugefügt. Die Verwendung von Headern ist für die eigentliche Anwendung reserviert.

Parameter	Eingangsdaten	Kabeldaten	Ausgangsdaten
`\Str`	1 char	1 byte (8 bits)	1 char
`\RawData`	1 rawbytes	1 byte (8 bits)	1 rawbytes
`\Data`	1 Byte	1 byte (8 bits)	1 Byte

In `SocketSendTo` und `SocketReceiveFrom` können verschiedene Datentypen gemeinsam verwendet werden (`string`, `rawbytes` oder `array of byte`).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_SOCK_CLOSED`	Der Socket ist geschlossen.
`ERR_SOCK_NET_UNREACH`	Das Netzwerk ist nicht erreichbar oder die Verbindung geht nach dem Öffnen eines Sockets verloren.
`ERR_SOCK_NOT_BOUND`	Der Socket wurde nicht an eine Adresse gebunden.
`ERR_SOCK_TIMEOUT`	Die Daten wurden nicht innerhalb der Timeout-Zeit empfangen.

Weitere Beispiele

Weitere Beispiele für die Instruktion `SocketReceiveFrom` sind unten aufgeführt.

Beispiel 1:

```
VAR socketdev udp_socket;
VAR string client_ip;
VAR num client_port;

PROC server_messaging()
  VAR string receive_string;
  ...
  ! Create and bind of sockets in error handlers
  SocketReceiveFrom udp_socket \Str := receive_string, client_ip,
       client_port;
  SocketSendTo udp_socket, client_ip, client_port \Str := "Hello
       client with ip-address"+client_ip;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  ...
  SocketClose udp_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=SOCK_CLOSED THEN
    messaging_recover;
    RETRY;
  ELSE
  ! No error recovery handling
  ENDIF
ENDPROC

PROC messaging_recover()
  SocketClose udp_socket;
  SocketCreate udp_socket \UDP;
  SocketBind udp_socket, "192.168.0.1", 1025;
ERROR
  IF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Dies ist ein Serverprogramm zum Erstellen und Binden von Sockets in Fehlerbehandlungen. Auf diese Weise kann das Programm einen Neustart nach einem Netzausfall behandeln.

In der Kommunikationsprozedur `server_messaging` empfängt der Server eine Zeichenfolgenmeldung vom Client und speichert die Meldung in `receive_string`. Anschließend antwortet der Server mit der Meldung `"Hello client with ip-address xxx.xxx.x.x"`.

Einschränkungen

Nach einem Neustart nach einem Netzausfall sind alle Sockets geschlossen. Dieses Problem kann von der Fehlerbehandlung behoben werden. Siehe obiges Beispiel.

Die maximale Datenmenge, die innerhalb eines Aufrufs empfangen werden kann, ist auf 1024 Byte beschränkt.

Syntax

```
SocketReceiveFrom
  [Socket ':='] <variable (VAR) of socketdev>
  ['\' Str ':=' <variable (VAR) of string>]
  |['\' RawData ':=' <variable (VAR) of rawbytes>]
  |['\' Data ':=' <array {*} (VAR) of byte>]
  ['\' ReadNoOfBytes ':=' <expression (IN) of num>]
  ['\' NoRecBytes ':=' <variable (VAR) of num>]
  [RemoteAddress ':='] <variable (VAR) of string>
  [RemotePort ':='] <variable (VAR) of num>
  ['\' Time ':=' <expression (IN) of num>]';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.252 SocketReceiveFrom - Daten vom Remote-Computer empfangen

Socket Messaging

Fortsetzung

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Senden von Daten an Remote-Computer	*SocketSendTo - Daten an Remote-Computer senden auf Seite 774*
Überprüfung eines Sockets auf vorhandene Daten hin.	*SocketPeek – Überprüfung eines Sockets auf vorhandene Daten hin auf Seite 1499*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.253 SocketSend - Senden von Daten an Remote-Computer

Verwendung

`SocketSend` wird verwendet, um Daten an einen Remote-Computer zu senden. `SocketSend` kann sowohl für Client- als auch für Serveranwendungen verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketSend`:

Siehe auch *Weitere Beispiele auf Seite 770*.

Beispiel 1:

```
SocketSend socket1 \Str := "Hello world";
```

Sendet die Meldung `"Hello world"` an den Remote-Computer.

Argumente

```
SocketSend Socket [ \Str ] | [ \RawData ] | [ \Data] [ \NoOfBytes
    ]
```

`Socket`

Datentyp: `socketdev`

In Client-Anwendungen muss der Socket, über den gesendet wird, bereits erstellt und verbunden sein.

In Server-Anwendungen muss der Socket, an den gesendet wird, bereits akzeptiert sein.

`[ \Str ]`

Datentyp: `string`

Die an den Remote-Computer zu sendende `string`.

`[ \RawData ]`

Datentyp: `rawbytes`

Die an den Remote-Computer zu sendenden `rawbytes`-Daten.

`[ \Data ]`

Datentyp: `array of byte`

Die Daten im an den Remote-Computer zu sendenden `byte`-Datenfeld.

Es kann jeweils nur einer der optionalen Parameter `\Str`, `\RawData` oder `\Data` verwendet werden.

`[ \NoOfBytes ]`

Datentyp: `num`

Wenn dieses Argument angegeben ist, wird nur diese Anzahl an Bytes an den Remote-Computer gesendet. Das Aufrufen von `SocketSend` schlägt fehl, wenn `\NoOfBytes` größer ist als die tatsächliche Anzahl an Bytes in der zu sendenden Datenstruktur.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.253 SocketSend - Senden von Daten an Remote-Computer
Socket Messaging
Fortsetzung

Wenn dieses Argument nicht angegeben ist, wird die gesamte Datenstruktur (der gültige Teil der `rawbytes`-Daten) an den Remote-Computer gesendet.

Programmabarbeitung

Die angegebenen Daten werden an den Remote-Computer gesendet. Falls die Verbindung unterbrochen wird, wird ein Fehler generiert.

Die über das Kabel übertragenen Daten sind immer byte-Daten. Eine Meldung kann aus maximal 1024 Byte bestehen. Der Meldung wird standardmäßig kein Header hinzugefügt. Die Verwendung von Headern ist für die eigentliche Anwendung reserviert.

Parameter	Eingangsdaten	Kabeldaten	Ausgangsdaten
`\Str`	1 char	1 byte (8 bits)	1 char
`\RawData`	1 rawbytes	1 byte (8 bits)	1 rawbytes
`\Data`	1 Byte	1 byte (8 bits)	1 Byte

In `SocketSend` und `SocketReceive` können verschiedene Datentypen gemeinsam verwendet werden (`string`, `rawbytes` oder `array of byte`).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_SOCK_CLOSED`	Der Socket ist geschlossen. Verbindung ist unterbrochen.
`ERR_SOCK_NET_UNREACH`	Das Netzwerk ist nicht erreichbar oder die Verbindung geht nach dem Öffnen eines Sockets verloren.
`ERR_SOCK_NOT_CONN`	Der Socket ist nicht verbunden.
`ERR_SOCK_UNSPEC`	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.

Weitere Beispiele

Weitere Beispiele für die Instruktion `SocketSend` sind unten aufgeführt.

Beispiel 1:

```
VAR socketdev client_socket;
VAR string receive_string;

PROC client_messaging()
  ...
  ! Create and connect the socket in error handlers
  SocketSend client_socket \Str := "Hello server";
  SocketReceive client_socket \Str := receive_string;
  ...
  SocketClose client_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    client_recover;
    RETRY;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC

PROC client_recover()
  SocketClose client_socket;
  SocketCreate client_socket;
  SocketConnect client_socket, "192.168.0.2", 1025;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Hierbei handelt es sich um ein Client-Programm zum Erstellen und Verbinden eines Sockets in Fehlerbehandlungen. Auf diese Weise kann das Programm einen Neustart nach einem Netzausfall behandeln.

In der Prozedur `client_recover` wird der Client-Socket erstellt und mit einem Remote-Computerserver mit der IP-Adresse `192.168.0.2` an Port `1025` verbunden.

Der Client sendet in der Kommunikationsprozedur `client_messaging` den Text `"Hello server"` an den Server und dieser antwortet dem Client mit `"Hello client"`. Diese Zeichenfolge wird in der Variablen `receive_string` gespeichert.

Beispiel 2:

```
VAR socketdev client_socket;
VAR string receive_string;

PROC client_messaging()
  ...
  ! Send cr and lf to the server
  SocketSend client_socket \Str := "\0D\0A";
  ...
ENDPROC
```

In diesem Beispiel sendet ein Client-Programm nicht druckbare Zeichen (Binärdaten) in einer Zeichenfolge. Dies kann bei der Kommunikation mit Sensoren oder anderen Clients, die solche Zeichen erfordert, sinnvoll sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.253 SocketSend - Senden von Daten an Remote-Computer

Socket Messaging

Fortsetzung

Einschränkungen

Die *Socket-Kommunikation* verfügt über keinen integrierten Synchronisierungsmechanismus, um den Empfang von Meldungen zu vermeiden, die aus mehreren gesendeten Meldungen bestehen. Es fällt in den Aufgabenbereich des Programmierers, die Synchronisierung durch *Bestätigungs*meldungen zu regeln (eine `SocketSend - SocketReceive`-Sequenz im Client- oder Serverprogramm muss vor nächsten `SocketSend - SocketReceive`-Sequenz abgeschlossen sein).

Nach einem Neustart nach einem Netzausfall sind alle Sockets geschlossen. Dieses Problem kann von der Fehlerbehandlung behoben werden. Siehe obiges Beispiel.

Vermeiden Sie schnelle Schleifen mit `SocketCreate ... SocketClose`, da der Socket nicht vor einem bestimmten Zeitpunkt geschlossen wird (TCP/IP-Funktionalität).

Die sendbare Datenmenge ist auf 1024 Byte beschränkt.

Syntax

```
SocketSend
  [Socket ':='] <variable (VAR) of socketdev>
  [\Str ':=' <expression (IN) of string>]
  |[\RawData ':=' <variable (VAR) of rawdata>]
  |[\Data ':=' <array {*} (IN) of byte>]
  ['\' NoOfBytes ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Verwendung von nicht druckbaren Zeichen (Binärdaten) in Zeichenfolgenliteralen	*Technical reference manual - RAPID kernel*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.254 SocketSendTo - Daten an Remote-Computer senden

Verwendung

`SocketSendTo` wird verwendet, um Daten an einen Remote-Computer zu senden. `SocketSendTo` kann sowohl für Client- als auch für Serveranwendungen verwendet werden.

`SocketSendTo` wird für verbindungslose Kommunikation mit UDP/IP verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SocketSendTo`:

Siehe auch *Weitere Beispiele auf Seite 770*.

Beispiel 1:

```
VAR socketdev udp_socket;

SocketCreate udp_socket \UDP;
SocketSendTo udp_socket, Address, Port \Str := "Hello world";
```

Sendet die Meldung `"Hello world"` an den Remote-Computer mit der IP-Adresse `Address` und dem Port `Port`.

Argumente

```
SocketSendTo Socket RemoteAddress RemotePort [ \Str ] | [ \RawData
     ] | [ \Data] [ \NoOfBytes ]
```

`Socket`

Datentyp: `socketdev`

Der Socket muss bereits erstellt worden sein.

`RemoteAddress`

Datentyp: `string`

Die Adresse des Remote-Computers. Der Remote-Computer muss durch eine IP-Adresse angegeben werden. Der Name des Remote-Computers kann nicht verwendet werden.

`RemotePort`

Datentyp: `num`

Der Port am Remote-Computer. Normalerweise stehen die Ports 1025-4999 zur freien Verfügung. Portnummern unter 1025 können bereits belegt sein.

`[ \Str ]`

Datentyp: `string`

Die an den Remote-Computer zu sendende `string`.

`[ \RawData ]`

Datentyp: `rawbytes`

Die an den Remote-Computer zu sendenden `rawbytes`-Daten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \Data ]`

Datentyp: `array of byte`

Die Daten im an den Remote-Computer zu sendenden `byte`-Datenfeld.

Es kann jeweils nur einer der optionalen Parameter `\Str`, `\RawData` oder `\Data` verwendet werden.

`[ \NoOfBytes ]`

Datentyp: `num`

Wenn dieses Argument angegeben ist, wird nur diese Anzahl an Bytes an den Remote-Computer gesendet. Das Aufrufen von `SocketSendTo` schlägt fehl, wenn `\NoOfBytes` größer ist als die tatsächliche Anzahl an Bytes in der zu sendenden Datenstruktur.

Wenn dieses Argument nicht angegeben ist, wird die gesamte Datenstruktur (der gültige Teil der `rawbytes`-Daten) an den Remote-Computer gesendet.

Programmabarbeitung

Die angegebenen Daten werden an den Remote-Computer gesendet.

Die über das Kabel übertragenen Daten sind immer byte-Daten. Eine Meldung kann aus maximal 1024 Byte bestehen. Der Meldung wird standardmäßig kein Header hinzugefügt. Die Verwendung von Headern ist für die eigentliche Anwendung reserviert.

Parameter	Eingangsdaten	Kabeldaten	Ausgangsdaten
`\Str`	1 char	1 byte (8 bits)	1 char
`\RawData`	1 rawbytes	1 byte (8 bits)	1 rawbytes
`\Data`	1 Byte	1 byte (8 bits)	1 Byte

In `SocketSendTo` und `SocketReceiveFrom` können verschiedene Datentypen gemeinsam verwendet werden (`string`, `rawbytes` oder `array of byte`).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_SOCK_CLOSED`	Der Socket ist geschlossen.
`ERR_SOCK_NET_UNREACH`	Das Netzwerk ist nicht erreichbar oder die Verbindung geht nach dem Öffnen eines Sockets verloren.

Weitere Beispiele

Weitere Beispiele für die Instruktion `SocketSendTo` sind unten aufgeführt.

Beispiel 1:

```
VAR socketdev client_socket;
VAR string receive_string;
VAR string RemoteAddress;
VAR num RemotePort;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.254 SocketSendTo - Daten an Remote-Computer senden

Socket Messaging

Fortsetzung

```
PROC client_messaging()
  ...
  ! Create and bind the socket in error handlers
  SocketSendTo client_socket, "192.168.0.2", 1025 \Str := "Hello
       server";
  SocketReceiveFrom client_socket \Str := receive_string,
       RemoteAddress, RemotePort;
  ...
  SocketClose client_socket;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    client_recover;
    RETRY;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC

PROC client_recover()
  SocketClose client_socket;
  SocketCreate client_socket \UDP;
  SocketBind client_socket, "192.168.0.2", 1025;
ERROR
  IF ERRNO=ERR_SOCK_TIMEOUT THEN
    RETRY;
  ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
    RETURN;
  ELSE
    ! No error recovery handling
  ENDIF
ENDPROC
```

Dies ist ein Client-Programm zum Erstellen und Binden von Sockets in Fehlerbehandlungen. Auf diese Weise kann das Programm einen Neustart nach einem Netzausfall behandeln.

In dem Verfahren `client_recover` wird der Client-Socket erstellt und mit einem Remote-Computerserver mit der IP-Adresse `192.168.0.2` mit Port `1025` verbunden.

Der Client sendet in der Kommunikationsprozedur `client_messaging` den Text `"Hello server"` an den Server und dieser antwortet dem Client mit `"Hello client"`. Diese Zeichenfolge wird in der Variablen `receive_string` gespeichert.

Beispiel 2:

```
VAR socketdev udp_socket;

PROC message_send()
  ...
  ! Send cr and lf to the server
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  SocketSendTo udp_socket, "192.168.0.2", 1025 \Str := "\0D\0A";
  ...
ENDPROC
```

In diesem Beispiel sendet ein Programm nicht druckbare Zeichen (Binärdaten) in einer Zeichenfolge. Dies kann bei der Kommunikation mit Sensoren oder anderen Clients, die solche Zeichen erfordern, sinnvoll sein.

Einschränkungen

Nach einem Neustart nach einem Netzausfall sind alle Sockets geschlossen. Dieses Problem kann von der Fehlerbehandlung behoben werden. Siehe obiges Beispiel.

Die sendbare Datenmenge ist auf 1024 Byte beschränkt.

Syntax

```
SocketSendTo
  [Socket ':='] <variable (VAR) of socketdev>
  [RemoteAddress ':='] <expression (IN) of string>
  [RemotePort ':='] <expression (IN) of num>
  [\Str ':=' <expression (IN) of string>]
  |[\RawData ':=' <variable (VAR) of rawdata>]
  |[\Data ':=' <array {*} (IN) of byte>]
  ['\' NoOfBytes ':=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Server-Socketanwendung	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Daten vom Remote-Computer empfangen	*SocketReceiveFrom - Daten vom Remote-Computer empfangen auf Seite 764*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.254 SocketSendTo - Daten an Remote-Computer senden

Socket Messaging

Fortsetzung

Information über	Siehe
Verwendung von nicht druckbaren Zeichen (Binärdaten) in Zeichenfolgenliteralen	*Technical reference manual - RAPID kernel*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.255 SoftAct - Aktivieren von Softservo

Verwendung

`SoftAct` (*Soft Servo Activate*) wird verwendet, um den so genannten Softservo auf einer beliebigen Achse des Roboters oder einer externen mechanischen Einheit zu aktivieren.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SoftAct`:

Beispiel 1:

```
SoftAct 3, 20;
```

Aktiviert Softservo an Roboterachse `3` mit Weichheitswert `20%`.

Beispiel 2:

```
SoftAct 1, 90 \Ramp:=150;
```

Aktiviert Softservo an Roboterachse `1` mit Weichheitswert `90%` und einer Rampe von `150%`.

Beispiel 3

```
SoftAct \MechUnit:=orbit1, 1, 40 \Ramp:=120;
```

Aktiviert Softservo an Achse `1` für die mechanische Einheit `orbit1` mit dem Weichheitswert `40%` und einer Rampe von `120%`.

Argumente

```
SoftAct [\MechUnit] Axis Softness [\Ramp]
```

`[ \MechUnit ]`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit Wenn dieses Argument ausgelassen wird, wird Softservo für die angegebene Roboterachse in der aktuellen Programmtask aktiviert.

`Axis`

Datentyp: `num`

Nummer der Roboter- oder der externen Achse, die mit Softservo arbeiten soll.

`Softness`

Datentyp: `num`

Weichheitswert in Prozent (0-100 %). 0 % bedeutet minimale Weichheit (maximale Starrheit) und 100 % bedeutet maximale Weichheit.

`[ \Ramp ]`

Datentyp: `num`

Rampe in Prozent (>= 100 %). Die Rampe steuert die Einstellung von Softservo. Ein Faktor von 100 % gibt den normalen Wert an. Mit größeren Werten wird der

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.255 SoftAct - Aktivieren von Softservo

RobotWare - OS

Fortsetzung

Softservo langsamer eingestellt (längere Rampe). Der Standardwert für die Rampe beträgt 100 %.

Programmabarbeitung

Die Weichheit wird mit dem Wert aktiviert, der für die aktuelle Achse angegeben ist. Der Weichheitswert gilt für alle Bewegungen, bis ein neuer Weichheitswert für die aktuelle Achse programmiert wird oder bis der Softservo durch die Instruktion `SoftDeact` deaktiviert wird.

Einschränkungen

Softservo wird immer für jede Roboter- oder externe Achse deaktiviert, wenn ein Stromausfall auftritt. Diese Einschränkung kann im Anwenderprogramm beim Neustart nach einem Stromausfall behandelt werden.

Dieselbe Achse darf nicht zweimal aktiviert sein, es sei denn, eine Bewegungsinstruktion wird dazwischen eingefügt. Daher sollte die folgende Programmsequenz vermieden werden, da sonst die Roboterbewegung ruckartig erfolgt:

```
SoftAct n , x ;
SoftAct n , y ;
```

(n = Roboterachse n, x und y = Weichheitswerte)

WARNUNG

Der Bremsweg von Stopps der Kategorie 1 wird länger, wenn der Softservomechanismus aktiv ist.

Syntax

```
SoftAct
  ['\'MechUnit ':=' < variable (VAR) of mecunit>',']
  [Axis ':=' ] < expression (IN) of num> ','
  [Softness':=' ] < expression (IN) of num> ','
  [ '\'Ramp':=' < expression (IN) of num> ]';'
```

Weitere Informationen

Information über	Siehe
Deaktivieren von Softservo	*SoftDeact - Deaktivieren von Softservo auf Seite 781*
Verhalten mit aktiviertem Softservo	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von externen Achsen	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.256 SoftDeact - Deaktivieren von Softservo

Verwendung

`SoftDeact` (*Soft Servo Deactivate*) deaktiviert die Funktion Softservo.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SoftDeact`:

Beispiel 1:

```
SoftDeact;
```

Deaktiviert Softservo auf allen Achsen.

Beispiel 2:

```
SoftDeact \Ramp:=150;
```

Deaktiviert Softservo auf allen Achsen mit Rampenfaktor 150 %.

Argumente

```
SoftDeact [\Ramp]
```

`[ \Ramp ]`

Datentyp: `num`

Rampe in Prozent (>= 100 %). Der Rampenfaktor wird verwendet, um die Deaktivierung von Softservo zu steuern. Ein Faktor von 100 % gibt den normalen Wert an. Mit größeren Werten wird der Softservo langsamer deaktiviert (längere Rampe). Der Standardwert für die Rampe beträgt 100 %.

Programmabarbeitung

Softservo wird für die mechanischen Einheiten deaktiviert, die von der aktuellen Programmtask gesteuert werden. Wenn `SoftDeact` von einer Nicht-Bewegungstask ausgeführt wird, wird Softservo für die von der verbundenen Bewegungstask gesteuerte mechanische Einheit deaktiviert. Durch Ausführen von `SoftDeact` im Synchronbewegungsmodus wird Softservo für alle synchronisierten mechanischen Einheiten deaktiviert.

Bei der Deaktivierung von Softservo über `SoftDeact` bewegt sich der Roboter an die programmierte Position. Dies ist selbst dann der Fall, wenn sich der Roboter bei der Aktivierung von Softservo von der Position entfernt hat.

Syntax

```
SoftDeact
  [ '\'Ramp ':=' < expression (IN) of num> ]';'
```

Weitere Informationen

Information über	Siehe
Aktivieren von Softservo	*SoftAct - Aktivieren von Softservo auf Seite 779*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.257 SoftElbow – Sorgt für Flexibilität des Ellenbogenstücks beim Einwirken von externen Kräften.

Verwendung

`SoftElbow` wird zum Aktivieren oder Deaktivieren von SoftElbow an einem Roboter mit 7 Achsen verwendet. Im aktiven Zustand kann der Ellenbogen so geschoben werden, dass die Ellenbogenposition geändert wird, ohne den TCP zu beeinflussen. Der TCP bewegt sich weiterhin auf seinem programmierten Weg.

Grundlegendes Beispiel

Die folgenden Beispiele veranschaulichen die Instruktion `SoftElbow`:

Beispiel 1:

```
SoftElbow \On;
```

Nach der Abarbeitung dieser Instruktion kann der Roboterellenbogen bewegt werden, ohne dabei das TCP zu beeinflussen.

Argumente

```
SoftElbow [\On] | [\Off]
```

`[\On]`

Datentyp: `switch`

Aktiviert SoftElbow.

`[\Off]`

Datentyp: `switch`

Deaktiviert SoftElbow.

Programmabarbeitung

Nach der Aktivierung von SoftElbow bleibt der Ellenbogen weich, bis er mit einer neuen `SoftElbow`-Anweisung ausgeschaltet wird.

Solange SoftElbow aktiv ist, beugt sich der Ellenbogen von jedem Hindernis oder allem, was auf den Ellenbogen drückt, weg. Wenn sich der Roboter bewegt, folgt der TCP weiterhin seinem Weg, die Programmausführung wird nicht unterbrochen.

Der Standardwert (SoftElbow deaktiviert) wird automatisch eingestellt:

- beim Verwenden des Neustart-Modus RAPID zurücksetzen
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Bewegen des Programmzeigers auf main (primär)
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

- `SoftElbow` ist nur für Roboter mit 7 Achsen verfügbar (z. B. Yumi).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.257 SoftElbow – Sorgt für Flexibilität des Ellenbogenstücks beim Einwirken von externen Kräften.

Seven axis robot
Fortsetzung

- `SoftElbow` funktioniert nicht, wenn das Werkzeug mit einem festen Gegenstand in Berührung kommt und eine erhebliche Kraft ausgeübt wird (z. B. beim Einpressen eines Gegenstandes in eine Halterung).
- Während der Arm geschoben wird, kann dies zu einer etwas geringeren Bahngenauigkeit führen.
- `SoftElbow` ist nicht kompatibel mit anderen konformen Modi (z. B. Lead Through, Force control oder SoftMove).
- `SoftElbow` ist nicht mit MultiMove Coordinated kompatibel.
- `SoftElbow` funktioniert optimal bei moderaten Geschwindigkeiten, typischerweise unter 1000 mm/s.
- Absolutgenauigkeit wird vorübergehend deaktiviert, während SoftElbow aktiv ist.

Syntax

```
SoftElbow
  ['\' On] | ['\' Off]';'
```


© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.258 SpeedLimAxis – Einstellen der Geschwindigkeitsbegrenzung für eine Achse

Verwendung

Mit `SpeedLimAxis` kann die Geschwindigkeitsgrenze für eine Achse festgelegt werden. Die Geschwindigkeitsbegrenzung erfolgt, wenn das Systemeingangssignal `LimitSpeed` auf 1 gesetzt wird. Mithilfe dieser Instruktion kann auch eine Geschwindigkeitsbegrenzung festgelegt werden, die erst später übernommen werden soll.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SpeedLimAxis`:

Beispiel 1:

```
SpeedLimAxis STN_1, 1, 20;
```

Damit wird die Geschwindigkeit der mechanischen Einheit `STN_1` an Achse 1 auf 20 Grad/Sekunde begrenzt, wenn der Systemeingang `LimitSpeed` auf 1 gesetzt ist.

Beispiel 2:

```
SpeedLimAxis ROB_1, 1, 10;
SpeedLimAxis ROB_1, 2, 30;
SpeedLimAxis ROB_1, 3, 30;
SpeedLimAxis ROB_1, 4, 30;
SpeedLimAxis ROB_1, 5, 30;
SpeedLimAxis ROB_1, 6, 30;
```

Hiermit wird die Geschwindigkeit an Achse 2 bis 6 auf 30 Grad/Sekunde und die Geschwindigkeit an Achse 1 für die mechanische Einheit `ROB_1` auf 10 Grad/Sekunde begrenzt, wenn der Systemeingang `LimitSpeed` auf 1 gesetzt ist.

Argumente

```
SpeedLimAxis MechUnit AxisNo AxisSpeed
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`AxisNo`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit.

`AxisSpeed`

Datentyp: `num`

Die Geschwindigkeit, die übernommen werden soll. Bei einer rotierenden Achse sollte die Geschwindigkeit in Grad/Sekunde und bei einer linearen Achse in mm/s angegeben sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Mit `SpeedLimAxis` wird die Geschwindigkeitsgrenze für eine Achse einer speziellen mechanischen Einheit eingestellt. Die Geschwindigkeit wird aber nicht sofort verringert, sondern es werden die Werte gespeichert und erst übernommen, sobald das Systemeingangssignal `LimitSpeed` auf 1 gesetzt wird.

Wird zur Grenzwerteinstellung einer Achse nicht `SpeedLimAxis` verwendet, wird stattdessen auch die Geschwindigkeitsbegrenzung für den Einrichtbetrieb genutzt. Falls für eine bestimmte Achse keine Begrenzung gewünscht wird, sollte ein großer Wert eingegeben werden. Wird weiterhin keine Geschwindigkeitsbeschränkung für den Kontrollpunkt mit der Instruktion `SpeedLimCheckPoint` gesetzt, kommt die Geschwindigkeitsbeschränkung für den Einrichtbetrieb zum Einsatz, um die Geschwindigkeit des Kontrollpunktes zu begrenzen.

Wenn das Systemeingangssignal `LimitSpeed` auf 1 gesetzt wird, fährt die Geschwindigkeit auf den gesenkten Wert herunter.

Wenn das Systemeingangssignal `LimitSpeed` auf 0 gesetzt wird, erhöht sich die Geschwindigkeit auf den in der aktuellen Bewegungsinstruktion programmierten Wert.

Die maximal zulässige Beschleunigung während des Hochfahrens wird durch die Systemparameter *Limit Speed Acc Limitation* in dem Typ *Motion Planner* gesteuert.

Das Systemausgangssignal `LimitSpeed` wird auf 1 gesetzt, sobald die reduzierte Geschwindigkeit erreicht wurde. Das Systemausgangssignal `LimitSpeed` wird auf 0 gesetzt, wenn die Geschwindigkeit anzusteigen beginnt.

Die Standardwerte für die Geschwindigkeitsbegrenzung werden automatisch eingestellt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AXIS_PAR`	Parameter "Achse" in der Instruktion ist falsch
`ERR_SPEEDLIM_VALUE`	Die im Argument `AxisSpeed` verwendete Geschwindigkeit ist zu niedrig.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.258 SpeedLimAxis – Einstellen der Geschwindigkeitsbegrenzung für eine Achse
RobotWare - OS
Fortsetzung

Weitere Beispiele

Weitere Beispiele für die Instruktion `SpeedLimAxis` **sind unten aufgeführt.**

Beispiel 1:

```
..
VAR intnum sigint1;
VAR intnum sigint2;
..
PROC main()
  ! Setup interrupts reacting on a signal input
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sigint1;
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sigint2;
  ..
  MoveL p1, z50, fine, tool2;
  MoveL p2, z50, fine, tool2;
  ..
  MoveL p10, v100, fine, tool2;
  ! Set limitations for checkpoints and axes
  SpeedLimCheckPoint 200;
  SpeedLimAxis ROB_1, 1, 10;
  SpeedLimAxis ROB_1, 2, 10;
  SpeedLimAxis ROB_1, 3, 10;
  SpeedLimAxis ROB_1, 4, 20;
  SpeedLimAxis ROB_1, 5, 20;
  SpeedLimAxis ROB_1, 6, 20;
  WHILE run_loop = TRUE DO
    MoveL p1, vmax, z50, tool2;
    ..
    MoveL p99, vmax, fine, tool2;
  ENDWHILE
  ! Set the default manual mode max speed
  SpeedLimCheckPoint 0;
  SpeedLimAxis ROB_1, 1, 0;
  SpeedLimAxis ROB_1, 2, 0;
  SpeedLimAxis ROB_1, 3, 0;
  SpeedLimAxis ROB_1, 4, 0;
  SpeedLimAxis ROB_1, 5, 0;
  SpeedLimAxis ROB_1, 6, 0;
  ..
TRAP setlimitspeed
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sig1int1;
  ! Set out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 1;
ENDTRAP
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
TRAP resetlimitspeed
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sig1int2;
  ! Reset out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 0;
ENDTRAP
```

Während der Roboterbewegung von Position `p1` zu `p10`, kommt die Standard-Geschwindigkeitsbeschränkung zum Einsatz (Geschwindigkeit im Einrichtbetrieb). Es wird eine neue Geschwindigkeitsbeschränkung für den TCP-Roboter und die Achsen hinzugefügt. Das `TRAP setlimitspeed` wendet die Geschwindigkeitsbeschränkung an, wenn das Signal `mysensorsignal` seinen Wert in 1 ändert.

Das `TRAP resetlimitspeed` entfernt die Geschwindigkeitsbeschränkung, wenn das Signal `mysensorsignal` seinen Wert in 0 ändert.

Diese neuen Einstellungen für die Geschwindigkeitsbeschränkung werden verwendet, solange die Variable `run_loop` auf `TRUE` gesetzt und das Systemeingangssignal `LimitSpeed` auf 1 eingestellt ist. Wenn `run_loop` auf `FALSE` gesetzt wird, wird die Standard-Höchstgeschwindigkeit (Einrichtbetrieb) eingestellt.

Hinweis

Die `TRAP`-Routine im Beispiel wird nur zur Visualisierung der Funktionalität genutzt. Das Signal für die Geschwindigkeitsbeschränkung kann auch entweder direkt mit dem Systemeingangssignal `LimitSpeed` oder über eine Sicherheits-SPS verbunden werden.

Einschränkungen

`SpeedLimAxis` kann nicht in der Ereignisroutine POWER ON verwendet werden.

Beim Reduzieren der Geschwindigkeit einer Achse oder eines Kontrollpunkts wird die Geschwindigkeit der anderen Achsen ebenfalls auf denselben Prozentsatz gesenkt, um auf derselben programmierten Bahn laufen zu können. Die Prozessgeschwindigkeit entlang der programmierten Bahn kann schwanken.

Bei der Verwendung von SafeMove zusammen mit der Geschwindigkeitsbegrenzung muss SafeMove eine Toleranz eingerichtet werden, da sich SafeMove und Bewegungskalkulation voneinander unterscheiden.

Syntax

```
SpeedLimAxis
  [MechUnit ':='] <variable (VAR) of mecunit>','
  [AxisNo ':='] <expression (IN) of num>','
  [AxisSpeed ':='] <expression (IN) of num>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.258 SpeedLimAxis – Einstellen der Geschwindigkeitsbegrenzung für eine Achse

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte	*SpeedLimCheckPoint – Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte auf Seite 789*
Systemeingangs- und -ausgangssignale	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.259 SpeedLimCheckPoint – Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte

Verwendung

Mit `SpeedLimCheckPoint` kann der Wert der Geschwindigkeitsbeschränkung für einen TCP-Roboter festgelegt werden. Die Geschwindigkeitsbeschränkung erfolgt, wenn das Systemeingangssignal `LimitSpeed` auf 1 gesetzt wird. Mithilfe dieser Instruktion kann auch eine Geschwindigkeitsbeschränkung gesetzt werden, die erst später übernommen werden soll.

Die Senkung der Geschwindigkeit erfolgt, wenn einer der Prüfpunkte den im `SpeedLimCheckPoint` festgelegten Prüfpunkt überschreitet. (Weitere Informationen zu Prüfpunkten siehe *Weitere Beispiele auf Seite 791*.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SpeedLimCheckPoint`:

Beispiel 1:

```
VAR num limit_speed:=200;
SpeedLimCheckPoint limit_speed;
```

Damit wird die Geschwindigkeit des TCP-Roboters auf 200 mm/s beschränkt, sobald der Systemeingang `LimitSpeed` auf 1 gesetzt wird.

Argumente

```
SpeedLimCheckPoint RobSpeed
```

`RobSpeed`

Datentyp: `num`

Die Geschwindigkeitsbeschränkung in mm/s, die übernommen werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

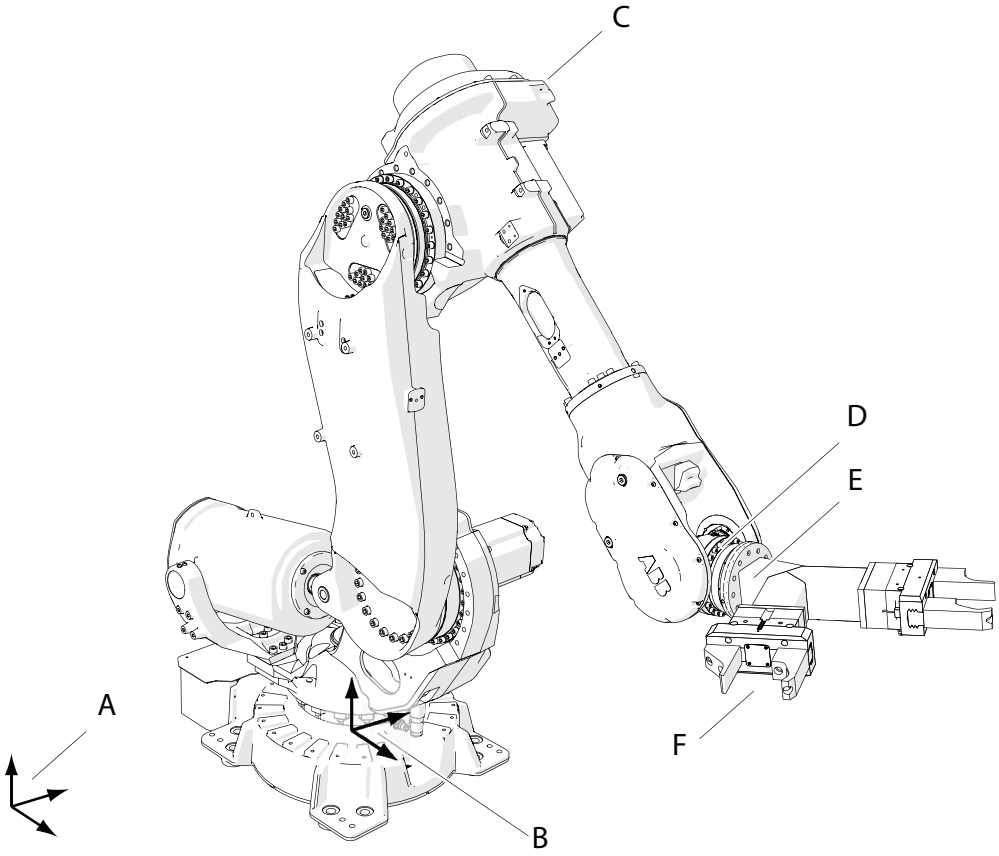
1.259 SpeedLimCheckPoint – Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte

RobotWare - OS

Fortsetzung

Programmabarbeitung

Definition der Prüfpunkte, siehe Abbildung unten.



xx1200000521

A	Welt-Koordinatensystem
B	Basis-Koordinatensystem
C	Arm-Prüfpunkt
D	Handgelenksmittelpunkt (WCP)
E	tool0
F	Werkzeugarbeitspunkt (TCP)

Mit `SpeedLimCheckPoint` kann eine Geschwindigkeitsbeschränkung für vier Prüfpunkte eines TCP-Roboters festgelegt werden. Die entsprechenden Prüfpunkte sind der Arm, das Handgelenk, tool0 und der aktive TCP, wie in der Abbildung oben zu sehen ist. Die Geschwindigkeit wird aber nicht sofort verringert, sondern es werden die Werte gespeichert und erst übernommen, sobald das Systemeingangssignal `LimitSpeed` auf 1 gesetzt wird. Die Geschwindigkeit an den Prüfpunkten wird im Verhältnis zum Basis-Koordinatensystem beschränkt.

Wird zur Grenzwerteinstellung nicht die Instruktion `SpeedLimCheckPoint` verwendet, wird stattdessen die Geschwindigkeitsbegrenzung für den Einrichtbetrieb genutzt. Falls für die Kontrollpunkte keine Begrenzung gewünscht wird, sollte ein großer Wert eingegeben werden. Wird weiterhin keine Geschwindigkeitsbeschränkung für die Achse mit der Instruktion `SpeedLimAxis`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

gesetzt, kommt die Geschwindigkeitsbeschränkung für den Einrichtbetrieb zum Einsatz, um die Geschwindigkeit der Achse zu begrenzen.

Wenn das Systemeingangssignal `LimitSpeed` auf 1 gesetzt wird, fährt die Geschwindigkeit auf den gesenkten Wert herunter.

Wenn das Systemeingangssignal `LimitSpeed` auf 0 gesetzt wird, erhöht sich die Geschwindigkeit auf den in der aktuellen Bewegungsinstruktion programmierten Wert.

Die maximal zulässige Beschleunigung während des Hochfahrens wird durch die Systemparameter *Limit Speed Acc Limitation* in dem Typ *Motion Planner* gesteuert.

Das Systemausgangssignal `LimitSpeed` wird auf 1 gesetzt, sobald die reduzierte Geschwindigkeit erreicht wurde. Das Systemausgangssignal `LimitSpeed` wird auf 0 gesetzt, wenn die Geschwindigkeit anzusteigen beginnt.

Die Standardwerte für die Geschwindigkeitsbegrenzung werden automatisch eingestellt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_SPEEDLIM_VALUE`	Die im Argument `RobSpeed` verwendete Geschwindigkeit ist zu niedrig.

Weitere Beispiele

Weitere Beispiele für die Instruktion `SpeedLimCheckPoint` sind unten aufgeführt.

Beispiel 1:

```
..
VAR intnum sigint1;
VAR intnum sigint2;
..
PROC main()
  ! Setup interrupts reacting on a signal input
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sigint1;
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sigint2;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.259 SpeedLimCheckPoint – Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte

RobotWare - OS

Fortsetzung

```
  ..
  MoveL p1, z50, fine, tool2;
  MoveL p2, z50, fine, tool2;
  ..
  MoveL p10, v100, fine, tool2;
  ! Set limitations for checkpoints and axes
  SpeedLimCheckPoint 200;
  SpeedLimAxis ROB_1, 1, 10;
  SpeedLimAxis ROB_1, 2, 10;
  SpeedLimAxis ROB_1, 3, 10;
  SpeedLimAxis ROB_1, 4, 20;
  SpeedLimAxis ROB_1, 5, 20;
  SpeedLimAxis ROB_1, 6, 20;
  WHILE run_loop = TRUE DO
    MoveL p1, vmax, z50, tool2;
    ..
    MoveL p99, vmax, fine, tool2;
  ENDWHILE
  ! Set the default manual mode max speed
  SpeedLimCheckPoint 0;
  SpeedLimAxis ROB_1, 1, 0;
  SpeedLimAxis ROB_1, 2, 0;
  SpeedLimAxis ROB_1, 3, 0;
  SpeedLimAxis ROB_1, 4, 0;
  SpeedLimAxis ROB_1, 5, 0;
  SpeedLimAxis ROB_1, 6, 0;
  ..
TRAP setlimitspeed
  IDelete sigint1;
  CONNECT sigint1 WITH setlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 1, sig1int1;
  ! Set out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 1;
ENDTRAP
TRAP resetlimitspeed
  IDelete sigint2;
  CONNECT sigint2 WITH resetlimitspeed;
  ISignalDI \SingleSafe, mysensorsignal, 0, sig1int2;
  ! Reset out signal that is cross connected to system input
       LimitSpeed
  SetDO do1LimitSpeed, 0;
ENDTRAP
```

Während der Roboterbewegung von Position `p1` zu `p10`, kommt die Standard-Geschwindigkeitsbeschränkung zum Einsatz (Geschwindigkeit im Einrichtbetrieb). Es wird eine neue Geschwindigkeitsbeschränkung für den TCP-Roboter und die Achsen hinzugefügt. Das `TRAP setlimitspeed` wendet die Geschwindigkeitsbeschränkung an, wenn das Signal `mysensorsignal` seinen Wert in 1 ändert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das TRAP `resetlimitspeed` entfernt die Geschwindigkeitsbeschränkung, wenn das Signal `mysensorsignal` seinen Wert in 0 ändert.

Diese neuen Einstellungen für die Geschwindigkeitsbeschränkung werden verwendet, solange die Variable `run_loop` auf `TRUE` gesetzt und das Systemeingangssignal `LimitSpeed` auf 1 eingestellt ist. Wenn `run_loop` auf `FALSE` gesetzt wird, wird die Standard-Höchstgeschwindigkeit (Einrichtbetrieb) eingestellt.

Hinweis

Die TRAP-Routine im Beispiel wird nur zur Visualisierung der Funktionalität genutzt. Das Signal für die Geschwindigkeitsbeschränkung kann auch entweder direkt mit dem Systemeingangssignal `LimitSpeed` oder über eine Sicherheits-SPS verbunden werden.

Einschränkungen

`SpeedLimCheckPoint` kann nicht in der Ereignisroutine POWER ON verwendet werden.

Wenn der Roboter auf einer sich bewegenden Verfahrachse steht, dann kann der Kontrollpunkt der Geschwindigkeit im Welt-Koordinatensystem höher sein, als der festgelegte Kontrollpunkt für die Geschwindigkeitsbeschränkung im Basis-Koordinatensystem. Der Kontrollpunkt der Geschwindigkeit im Welt-Koordinatensystem kann die Summe der Geschwindigkeit der Verfahrachse und der Geschwindigkeit des Kontrollpunkts im Basis-Koordinatensystem sein. Um die Geschwindigkeit des Kontrollpunktes im Welt-Koordinatensystem zu begrenzen, stellen Sie sicher, dass beide die Geschwindigkeitsbeschränkung nicht überschreiten.

Beim Reduzieren der Geschwindigkeit einer Achse oder eines Kontrollpunkts wird die Geschwindigkeit der anderen Achsen ebenfalls auf denselben Prozentsatz gesenkt, um auf derselben programmierten Bahn laufen zu können. Die Prozessgeschwindigkeit entlang der programmierten Bahn kann schwanken.

Bei der Verwendung von SafeMove zusammen mit der Geschwindigkeitsbegrenzung, muss SafeMove mit einer Toleranz eingerichtet werden und diese getestet werden, da sich SafeMove und Bewegungskalkulation voneinander unterscheiden. Ein spontaner Wechsel des Werkzeug-TCP ist nicht synchronisiert mit SafeMove. Deswegen muss der Werkzeug-TCP SafeMove entweder kürzer sein als die Werkzeuge, die vom Roboter benutzt werden oder die maximale Geschwindigkeit des Kontrollpunkts für SafeMove muss mit einer zusätzlichen Toleranzgrenze eingerichtet und getestet werden.

Syntax

```
SpeedLimCheckPoint
  [RobSpeed ':='] <expression (IN) of num>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.259 SpeedLimCheckPoint – Einstellen der Geschwindigkeitsbeschränkung für Prüfpunkte

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Einstellen der Geschwindigkeitsbegrenzung für eine Achse	*SpeedLimAxis – Einstellen der Geschwindigkeitsbegrenzung für eine Achse auf Seite 784*
Definieren der Armlasten	*Technisches Referenzhandbuch - Systemparameter*
Systemeingangs- und -ausgangssignale	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.260 SpeedRefresh – Geschwindigkeitsübersteuerung für laufende Bewegung aktualisieren

Verwendung

`SpeedRefresh` wird zum Ändern der Bewegungsgeschwindigkeit für die gegenwärtig ausgeführte Roboterbewegung in der aktuellen Bewegungsprogrammtask verwendet. Mit dieser Instruktion kann eine grobe Geschwindigkeitsanpassung anhand einer Sensoreingabe erstellt werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SpeedRefresh`:

Beispiel 1:

```
VAR num change_speed:=70;
SpeedRefresh change_speed;
```

Hierdurch wird der aktuelle Geschwindigkeitsoverride in 70 % geändert.

Argumente

```
SpeedRefresh Override
```

`Override`

Datentyp: `num`

Der Geschwindigkeitsoverride-Wert im Bereich 0 ... 100 %.

Programmabarbeitung

Der gegenwärtige Geschwindigkeitsoverride-Wert für gegenwärtige Bewegungen von Roboter und externen Einheiten in der aktuellen Bewegungsprogrammtask wird aktualisiert.

Dies wirkt sich auf alle Geschwindigkeitsdatenkomponenten für jegliche mechanischen Einheiten in der aktuellen Bewegungstask aus.

Hinweis

Die von `SpeedRefresh` eingestellte Geschwindigkeitsübersteuerung ist nicht gleichwertig mit der Geschwindigkeitseinstellung vom FlexPendant. Dies sind zwei verschiedene Werte. Das Produkt dieser zwei Werte und die programmierte Geschwindigkeit stellen die Geschwindigkeit dar, die für die Bewegung verwendet wird.

Wenn der PZ in die Hauptstellung gesetzt (PP to main) oder ein neues Programm geladen wird, wird die mit `SpeedRefresh` eingestellte Geschwindigkeit zurückgesetzt, und die auf dem FlexPendant festgelegte Geschwindigkeit wird angewendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.260 SpeedRefresh – Geschwindigkeitsübersteuerung für laufende Bewegung aktualisieren
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SPEED_REFRESH_LIM	Override hat einen Wert außerhalb des Bereichs von 0 bis 100 %.

Weitere Beispiele

Weitere Beispiele für die Instruktion SpeedRefresh sind unten aufgeführt.

Beispiel 1:

```
VAR intnum time_int;
VAR num override;
...
PROC main()
  CONNECT time_int WITH speed_refresh;
  ITimer 0.1, time_int;
  ISleep time_int;
  ...
  MoveL p1, v100, fine, tool2;
  ! Read current speed override set from FlexPendant
  override := CSpeedOverride (\CTask);
  IWatch time_int;
  MoveL p2, v100, fine, tool2;
  IDelete time_int;
  ! Reset to FlexPendant old speed override
  WaitTime 0.5;
  SpeedRefresh override;
  ...
TRAP speed_refresh
  VAR speed_corr;
  ! Analog input signal value from sensor, value 0 ... 10
  speed_corr := (ai_sensor * 10);
  SpeedRefresh speed_corr;
  ERROR
    IF ERRNO = ERR_SPEED_REFRESH_LIM THEN
      IF speed_corr > 100 speed_corr := 100;
      IF speed_corr < 0 speed_corr := 0;
      RETRY;
    ENDIF
ENDTRAP
```

Während der Roboterbewegung von Position p1 zu Position p2wird der Geschwindigkeitsoverride-Wert alle 0.1 s in der TRAP-Routine speed_refresh aktualisiert. Mit dem analogen Eingangssignal ai_sensor wird derOveride -Wert für die Instruktion SpeedRefresh berechnet. Es erfolgt keine TRAP-Abarbeitung vor und nach der Roboterbewegung zwischen p1 und p2. Der manuelle Geschwindigkeitsoverride vom FlexPendant wird wiederhergestellt. Anschließend muss der Roboter p2 erreichen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.260 SpeedRefresh – Geschwindigkeitsübersteuerung für laufende Bewegung aktualisieren

Einschränkungen

Beachten Sie, dass mit `SpeedRefresh` die Geschwindigkeitsübersteuerung nicht sofort ausgeführt wird. Stattdessen wird der Befehl mit einer Verzögerung von 0,3 - 0,5 Sekunden am physischen Roboter wirksam.

Der Geschwindigkeitsoverride-Wert vom RAPID-Programm muss vom Benutzer nach der `SpeedRefresh`-Sequenz zurückgesetzt werden.

Wenn `SpeedRefresh` in der Ereignisroutine `START` oder `RESET` verwendet wird, handelt es sich bei der gesetzten Geschwindigkeit immer um die aktuelle FlexPendant-Geschwindigkeitsübersteuerung.

Syntax

```
SpeedRefresh
  [ Override ':=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Lesen des aktuellen Geschwindigkeitsoverrides	*CSpeedOverride - Lesen der aktuellen Overridegeschwindigkeit auf Seite 1264*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.261 SpyStart - Starten der Aufzeichnung von Abarbeitungsdaten

Verwendung

`SpyStart` wird verwendet, um die Aufzeichnung von Instruktions- und Zeitdaten während der Abarbeitung zu starten.

Die Abarbeitungsdaten werden zur späteren Analyse in einer Datei gespeichert.

Die gespeicherten Daten sind zur Fehlersuche in RAPID-Programmen gedacht, insbesondere für Multitasking-Systeme (`SpyStart - SpyStop` ist nur in einer Programmtask erforderlich).

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SpyStart`:

Beispiel 1:

```
SpyStart "HOME:/spy.log";
```

Startet die Aufzeichnung der Abarbeitungsdaten in der Datei `spy.log` im `HOME`:-Verzeichnis.

Argumente

```
SpyStart File
```

Datei

Datentyp: `string`

Pfad und Name der Datei, die die Abarbeitungsdaten enthalten wird.

Programmabarbeitung

Die angegebene Datei wird zum Schreiben geöffnet und die Abarbeitungsdaten werden in der Datei aufgezeichnet.

Die Aufzeichnung der Abarbeitungsdaten ist aktiv bis:

- Ausführung der Instruktion `SpyStop`
- Start der Programmabarbeitung von Anfang an,
- Laden eines neues Programms,
- Weiter **Neustart**
- Umschalten von manuell auf automatisch und Auto Condition Reset aktiviert

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEOPEN`	Die in der Instruktion `SpyStart` angegebene Datei kann nicht geöffnet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dateiformat

TASK	INSTR	IN	CODE	OUT
MAIN	FOR i FROM 1 TO 3 DO	0	READY	0
MAIN	mynum:=mynum+i;	1	READY	1
MAIN	ENDFOR	2	READY	2
MAIN	mynum:=mynum+i;	2	READY	2
MAIN	ENDFOR	2	READY	2
MAIN	mynum:=mynum+i;	2	READY	2
MAIN	ENDFOR	2	READY	3
MAIN	SetDo1,1;	3	READY	3
MAIN	IF di1=0 THEN	3	READY	4
MAIN	MoveL p1, v1000, fine, tool0;	4	WAIT	14
MAIN	MoveL p1, v1000, fine, tool0;	111	READY	111
MAIN	ENDIF	108	READY	108
MAIN	MoveL p2, v1000, fine, tool0;	111	WAIT	118
MAIN	MoveL p2, v1000, fine, tool0;	326	READY	326
MAIN	SpyStop;	326	READY	

Die Spalte TASK enthält die abgearbeitete Programmtask.

Die Spalte INSTR enthält die abgearbeitete Instruktion in der angegebenen Programmtask.

Die Spalte IN enthält die Zeit in Millisekunden zu Beginn der abgearbeiteten Instruktion.

Die Spalte CODE zeigt an, ob die Instruktion fertig abgearbeitet (READY) ist oder zur angegebenen OUT-Zeit auf Beendigung wartet (WAIT).

Die Spalte OUT enthält die Zeit in Millisekunden bei Beendigung der abgearbeiteten Instruktion.

Alle Zeiten werden in Millisekunden (ms) angegeben.

SYSTEM TRAP gibt an, dass das System andere Aktionen ausführt und keine RAPID-Instruktionen abarbeitet.

Beim Aufruf einer NOSTEPIN-Prozedur (Modul) zeigt die ausgegebene Liste nur den Namen der aufgerufenen Prozedur an. Dies wird für jede abgearbeitete Instruktion in der Routine NOSTEPIN wiederholt.

Einschränkungen

Verwenden Sie diese Funktionalität nicht während der Produktion, da sie die Zyklusdauer erhöht und Speicherplatz auf dem benutzten Massenspeicher belegt.

Syntax

```
SpyStart
  [File':=']<expression (IN) of string>';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.261 SpyStart - Starten der Aufzeichnung von Abarbeitungsdaten

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Stoppen der Aufzeichnung von Abarbeitungsdaten	*SpyStop - Stoppen der Aufzeichnung von Abarbeitungsdaten auf Seite 801*
Auto Condition	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.262 SpyStop - Stoppen der Aufzeichnung von Abarbeitungsdaten

Verwendung

`SpyStop` wird verwendet, um die Aufzeichnung von Zeitdaten während der Abarbeitung zu stoppen.

Die Daten, die nützlich für die Optimierung der Abarbeitungszyklusdauer sein können, werden zur späteren Analyse in einer Datei gespeichert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SpyStop` :

Siehe auch *Weitere Beispiele auf Seite 801*.

Beispiel 1:

```
SpyStop;
```

Stoppt die Aufzeichnung der Abarbeitungsdaten in der Datei, die durch die vorherige Instruktion `SpyStart` angegeben wurde.

Programmabarbeitung

Die Aufzeichnung der Abarbeitungsdaten wird gestoppt und die Datei, die durch die vorherige Instruktion `SpyStart` angegeben wurde, wird geschlossen. Wenn vorher keine `SpyStart`-Instruktion abgearbeitet wurde, wird die Instruktion `SpyStop` ignoriert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `SpyStop` sind unten aufgeführt.

Beispiel 1:

```
IF debug = TRUE SpyStart "HOME:/spy.log";
produce_sheets;
IF debug = TRUE SpyStop;
```

Wenn das Debug-Flag TRUE ist, beginnt die Aufzeichnung der Abarbeitungsdaten in der Datei `spy.log` im Verzeichnis `HOME`: Der Produktionszyklus wird ausgeführt, dann wird die Aufzeichnung gestoppt und die Datei `spy.log` wird geschlossen.

Einschränkungen

Verwenden Sie diese Funktionalität nicht während der Produktion, da sie die Zyklusdauer erhöht und Speicherplatz auf dem benutzten Massenspeicher belegt.

Syntax

```
SpyStop';'
```

Weitere Informationen

Information über	Siehe
Starten der Aufzeichnung von Abarbeitungsdaten	*SpyStart - Starten der Aufzeichnung von Abarbeitungsdaten auf Seite 798*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.263 StartLoad – Laden eines Programmmoduls während der Abarbeitung

Verwendung

`StartLoad` wird verwendet, um während der Abarbeitung das Laden eines Programmmoduls in den Programmspeicher zu starten.

Wenn der Ladevorgang läuft, können andere Instruktionen parallel ausgeführt werden. Das geladene Modul muss mit der Instruktion `WaitLoad` mit der Programmtask verbunden werden, bevor seine Symbole/Routinen benutzt werden können.

Das geladene Programmmodul wird den bereits bestehenden Modulen im Programmspeicher hinzugefügt.

Ein Programm- oder Systemmodul kann im statischen (Standard) oder dynamischen Modus geladen werden. Abhängig vom verwendeten Modus entladen einige Operationen das Modul oder wirken sich überhaupt nicht auf das Modul aus.

Statischer Modus

Folgende Tabelle zeigt, wie verschiedene Operationen statisch geladene Programm- oder Systemmodule beeinflussen.

	PZ vom PG aus auf "main" setzen	Neues RAPID-Programm öffnen
Programmmodul	Kein Einfluss	Entfernen
Systemmodul	Kein Einfluss	Kein Einfluss

Dynamischer Modus

Folgende Tabelle zeigt, wie verschiedene Operationen dynamisch geladene Programm- oder Systemmodule beeinflussen.

	PZ vom PG aus auf "main" setzen	Neues RAPID-Programm öffnen
Programmmodul	Entfernen	Entfernen
Systemmodul	Entfernen	Entfernen

Sowohl statisch als auch dynamisch geladene Module können durch die Instruktion `UnLoad` entfernt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `StartLoad`:

Siehe auch *Weitere Beispiele auf Seite 804*.

Beispiel 1:

```
VAR loadsession load1;

! Start loading of new program module PART_B containing routine
     routine_b in dynamic mode
StartLoad \Dynamic, diskhome \File:="PART_B.MOD", load1;

! Executing in parallel in old module PART_A containing routine_a
%"routine_a"%;

! Unload of old program module PART_A
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
UnLoad diskhome \File:="PART_A.MOD";
! Wait until loading and linking of new program module PART_B is
     ready
WaitLoad load1;

! Execution in new program module PART_B
%"routine_b"%;
```

Startet mit der Instruktion `StartLoad` das Laden des Programmmoduls `PART_B.MOD` von `diskhome` in den Programmspeicher. Parallel mit dem Ladevorgang arbeitet das Programm `routine_a` in Modul `PART_A.MOD` ab. Dann wartet die Instruktion `WaitLoad`, bis der Lade- und Verbindungsvorgang abgeschlossen ist. Das Modul wird im dynamischen Modus geladen.

Die Variable `load1` enthält die Identität des Ladevorgangs, aktualisiert durch `StartLoad` und referenziert durch `WaitLoad`.

Um Verbindungszeit zu sparen, können die Instruktionen `UnLoad` und `WaitLoad` in der Instruktion `WaitLoad` kombiniert werden, und zwar mithilfe des optionalen Arguments `\UnLoadPath`.

Argumente

```
StartLoad [\Dynamic] FilePath [\File] LoadNo
```

`[\Dynamic]`

Datentyp: `switch`

Der Schalter (switch) ermöglicht das dynamische Laden eines Programmmoduls. Andernfalls erfolgt das Laden im statischen Modus.

`FilePath`

Datentyp: `string`

Pfad und Name der Datei, die in den Programmspeicher geladen wird. Der Dateiname soll ausgeschlossen werden, wenn das Argument `\File` verwendet wird.

`[\File]`

Datentyp: `string`

Wenn der Dateiname im Argument `FilePath` ausgelassen wird, muss er mit diesem Argument definiert werden.

`LoadNo`

Datentyp: `loadsession`

Dies ist ein Verweis auf den Ladevorgang, der in der Instruktion `WaitLoad` verwendet werden muss, um das geladene Programmmodul mit der Programmtask zu verbinden.

Programmabarbeitung

Die Abarbeitung von `StartLoad` fordert den Ladevorgang nur an und fährt dann direkt mit der nächsten Instruktion fort, ohne den Abschluss des Ladevorgangs abzuwarten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.263 StartLoad – Laden eines Programmmoduls während der Abarbeitung

RobotWare - OS

Fortsetzung

Die Instruktion `WaitLoad` wartet dann auf das Ende des Ladevorgangs (sofern noch nicht beendet), bevor die Verbindung und Initialisierung stattfindet. Die Initialisierung des geladenen Moduls setzt alle Variablen auf Modulebene auf ihre Anfangswerte.

Nicht aufgelöste Verweise werden für den Ladevorgang `StartLoad - WaitLoad` standardmäßig akzeptiert, doch bei der Abarbeitung eines nicht aufgelösten Verweises tritt ein Laufzeitfehler auf.

Für eine gute Programmstruktur, die sich leicht verstehen und pflegen lässt, sollten sämtliche Lade- und Entladevorgänge von Programmmodulen im Hauptmodul erfolgen, das stets bei der Abarbeitung im Programmspeicher vorhanden ist.

Informationen über das Laden eines Programms, das eine `main`-Prozedur zu einem Hauptprogramm (mit einer weiteren `main`-Prozedur) enthält, finden Sie unter der Instruktion `Load`, *Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346*.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILNOTFND`	Datei nicht gefunden.
`ERR_LOADNO_INUSE`	Die im Argument `LoadNo` angegebene Variable wird bereits verwendet.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `StartLoad` sind unten aufgeführt.

Beispiel 1:

```
StartLoad \Dynamic, "HOME:/DOORDIR/DOOR1.MOD", load1;
```

Lädt das Programmmodul `DOOR1.MOD` aus `HOME:` aus dem Verzeichnis `DOORDIR` in den Programmspeicher. Das Programmmodul wird im dynamischen Modus geladen.

Beispiel 2:

```
StartLoad \Dynamic, "HOME:" \File:="/DOORDIR/DOOR1.MOD", load1;
```

Wie in Beispiel 1, aber mit unterschiedlicher Syntax.

Beispiel 3

```
StartLoad "HOME:" \File:="/DOORDIR/DOOR1.MOD", load1;
```

Wie in den obigen Beispielen 1 und 2, aber das Modul wird im statischen Modus geladen.

Beispiel 4:

```
StartLoad \Dynamic, "HOME:" \File:="/DOORDIR/DOOR1.MOD", load1;
WaitLoad load1;
```

ist identisch mit

```
Load \Dynamic, "HOME:" \File:="/DOORDIR/DOOR1.MOD";
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Bei einem Stromausfall während der Ausführung einer `StartLoad`-Instruktion kann das System bei Rückkehr der Versorgung nicht starten. Möglicherweise geht die Steuerung in einen Systemfehler-Status über. Dann muss das System aus einem Backup wiederhergestellt werden.

Um dieses Risiko gering zu halten, ist es besser mehrere kleine Module anstelle eines großen Moduls zu laden.

Syntax

```
StartLoad
  ['\'Dynamic ',']
  [FilePath' :='] <expression (IN) of string>
  ['\'File ':=' <expression (IN) of string> ] ','
  [LoadNo ':='] <variable (VAR) of loadsession>';'
```

Weitere Informationen

Information über	Siehe
Verbinden des geladenen Moduls mit der Task	*WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*
Ladevorgang	*loadsession – Ladevorgang für ein Programm auf Seite 1730*
Laden eines Programmmoduls	*Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346*
Entfernen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*
Abbruch des Ladens eines Programmmoduls	*CancelLoad - Abbruch des Ladens eines Moduls auf Seite 81*
Prozeduraufruf mit später Bindung	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.264 StartMove - Erneutes Starten der Roboterbewegung

Verwendung

`StartMove` wird verwendet, um die Bewegung der Roboter- und externen Achsen sowie des zugehörigen Prozesses fortzusetzen, nachdem die Bewegung angehalten wurde.

- mit der Instruktion `StopMove`.
- nach der Abarbeitung der `StorePath` ... `RestoPath`-Sequenz.
- nach asynchron ausgelösten Bewegungsfehlern, z. B. `ERR_PATH_STOP`, oder nach bestimmten Prozessfehlern nach der Behandlung in der `ERROR`-Behandlungsroutine

Im Basissystem kann diese Instruktion in folgenden Programmtasktypen verwendet werden:

- in der Haupttask `T_ROB1`, für die Wiederaufnahme der Bewegung in dieser Task.
- in einer beliebigen anderen Task zur Wiederaufnahme der Bewegungen in der Haupttask.

In einem MultiMove-System kann diese Instruktion in folgenden Programmtasktypen verwendet werden:

- in der Bewegungstask zur Wiederaufnahme der Bewegung in dieser Task.
- in einer Task, die keine Bewegungstask ist, zur Wiederaufnahme der Bewegung in der verbundenen Bewegungstask. Bei Wiederaufnahme der Bewegung in einer Bewegungstask aus einer koordinierten synchronisierten Taskgruppe wird zusätzlich die Bewegung in allen koordinierten Tasks wiederaufgenommen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `StartMove`:

Beispiel 1:

```
StopMove;
WaitDI ready_input,1;
StartMove;
```

Die Roboterbewegung wird wieder gestartet, wenn der Eingang `ready_input` gesetzt wird.

Beispiel 2:

```
...
MoveL p100, v100, z10, tool1;
StorePath;
p:= CRobT(\Tool:=tool1);
! New temporary movement
MoveL p1, v100, fine, tool1;
...
MoveL p, v100, fine, tool1;
RestoPath;
StartMove;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
...
```

Der Roboter wird nach dem Zurückbewegen zu Stoppposition `p` (in diesem Beispiel `p100`) gestartet, um die Bewegung auf der Basisbahnebene fortzusetzen.

Argumente

```
StartMove [\AllMotionTasks]
```

`[\AllMotionTasks]`

Datentyp: `switch`

Wiederaufnahme der Bewegung in allen mechanischen Einheiten im System. Der Schalter `[\AllMotionTasks]` kann nur aus einer Programmtask verwendet werden, die keine Bewegungstask ist.

Programmabarbeitung

Alle mit der gestoppten Bewegung verbundenen Prozesse werden gleichzeitig mit der Wiederaufnahme der Bewegung fortgesetzt.

Hinweis

Wenn eine `StopMove`-Anweisung für das Stoppen der Bewegung verwendet wurde, dann muss eine `StartMove`-Anweisung in derselben Task ausgeführt werden, die den `StopMove` ausführte. Dieses Verhalten ist unabhängig davon, ob das Argument `\AllMotionTasks` verwendet wurde oder nicht identisch.

Um eine MultiMove-Anwendung im synchronisierten koordinierten Modus neu zu starten, muss `StartMove` in allen Bewegungstasks ausgeführt werden, die koordiniert werden.

Mit dem Schalter `\AllMotionTasks` (nur zulässig in Tasks, die keine Bewegungstasks sind) werden die Bewegungen für alle mechanischen Einheiten im System wiederaufgenommen.

In einem Basissystem ohne den Schalter `\AllMotionTasks` werden die Bewegungen der folgenden mechanischen Einheiten wiederaufgenommen:

- immer die der mechanischen Einheiten der Haupttask, unabhängig davon, welche Task die Instruktion `StartMove` ausführt.

In einem MultiMove-System ohne den Schalter `\AllMotionTasks` werden die Bewegungen der folgenden mechanischen Einheiten wiederaufgenommen:

- die der mechanischen Einheiten in der Bewegungstask, die die Instruktion `StartMove` ausführen.
- die mechanischen Einheiten in der Bewegungstask, die mit der Task verbunden sind, die keine Bewegungstask ist und die Instruktion `StartMove` ausführt. Bei Neustart der mechanischen Einheiten in einer der verbundenen Bewegungstasks aus einer koordinierten synchronisierten Taskgruppe werden zusätzlich alle mechanischen Einheiten in den koordinierten Tasks neu gestartet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.264 StartMove - Erneutes Starten der Roboterbewegung

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_PATHDIST	Der Roboter ist für einen Neustart der unterbrochenen Bewegung zu weit von der Bahn entfernt (mehr als 10 mm oder 20 Grad). Der Roboter muss vor der Ausführung von RETRY näher an die Bahn bewegt werden.
ERR_STARTMOVE	Der Roboter befindet sich bei Ausführung der Instruktion StartMove im Haltezustand. Vor RETRY muss einige Zeit gewartet werden.
ERR_PROGSTOP	Der Roboter befindet sich bei Ausführung der Instruktion StartMove im Zustand Programmstopp. Vor RETRY muss einige Zeit gewartet werden.
ERR_ALRDY_MOVING	Der Roboter bewegt sich bereits bei der Abarbeitung der Instruktion StartMove . Vor RETRY muss einige Zeit gewartet werden.

Einschränkungen

Um die StopMove - StartMove-Sequenz für eine Bewegungstask auszuführen, ist nur eine von mehreren Nicht-Bewegungstasks gleichzeitig zulässig.

Die Fehlerbehebung ist nicht möglich, wenn StartMove in einer Fehlerbehandlungsroutine ausgeführt wird.

Syntax

```
StartMove
  ['\'AllMotionTasks]';'
```

Weitere Informationen

Information über	Siehe
Stoppen von Bewegungen	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Fortfahren mit einer Bewegung	*StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung auf Seite 809*
Weitere Beispiele	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.265 StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung

Verwendung

`StartMoveRetry` wird zur Wiederaufnahme der Bewegungen und des Zuordnungsprozesses der Roboterachsen und der externen Achsen und außerdem für den erneuten Abarbeitungsversuch über eine `ERROR`-Behandlungsroutine verwendet.

Diese Instruktion kann in einer `ERROR`-Behandlungsroutine in folgenden Typen von Programmtasks verwendet werden:

- in einer Haupttask `T_ROB1` in einem Basissystem
- in einer beliebigen Bewegungstask in einem *MultiMove*-System

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `StartMoveRetry`:

Beispiel 1:

```
VAR robtarget p_err;
...
MoveL p1\ID:=50, v1000, z30, tool1 \WObj:=stn1;
...
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StorePath;
    p_err := CRobT(\Tool:= tool1 \WObj:=wobj0);
    ! Fix the problem
    MoveL p_err, v100, fine, tool1;
    RestoPath;
    StartMoveRetry;
  ENDIF
ENDPROC
```

Hierbei handelt es sich um ein Beispiel aus einem MultiMove-System mit koordinierten synchronisierten Bewegungen (zwei Roboter, die an einem gedrehten Werkstück arbeiten).

Bei der Bewegung zu Position `p1` stößt der andere koordinierte Roboter auf einen Prozessfehler, so dass die koordinierten synchronisierten Bewegungen angehalten werden. Die Roboter erhalten nun die Fehlermeldung `ERR_PATH_STOP` und die Abarbeitung wird an die `ERROR`-Behandlungsroutine übergeben.

In der `ERROR`-Behandlungsroutine geschieht Folgendes:

- `StorePath` speichert die ursprüngliche Bahn, geht auf eine neue Bahnebene und versetzt das MultiMove-System in den unabhängigen Modus.
- Wenn Probleme mit dem Roboter auftreten, werden Bewegungen auf der neuen Bahnebene initiiert.
- Vor `RestoPath` wird zur Fehlerposition zurückgekehrt.
- Mit `RestoPath` kehrt der Roboter auf die ursprüngliche Bahnebene zurück und das MultiMove-System wird wieder in den synchronisierten Modus gesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.265 StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung
RobotWare - OS
Fortsetzung

- `StartMoveRetry` startet die unterbrochene Bewegung und unterbrochene Prozesse neu. Mit dieser Instruktion wird außerdem die Abarbeitung auf die Fortsetzung der normalen Abarbeitung zurückgesetzt.

Programmabarbeitung

`StartMoveRetry` löst folgende Sequenz aus:

- Rückkehr zur Bahn
- Neustart aller Prozesse, die bei den unterbrochenen Bewegungen ablaufen
- Neustart der unterbrochenen Bewegungen
- `RETRY` der Programmabarbeitung

In `StartMoveRetry` werden die Vorgänge von `StartMove` und `RETRY` in einem einzigen, nicht aufteilbaren Vorgang gemeinsam ausgeführt.

Es werden nur die mechanischen Einheiten in der Programmtask erneut gestartet, die `StartMoveRetry` ausführen.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_PATHDIST`	Der Roboter ist für einen Neustart der unterbrochenen Bewegung zu weit von der Bahn entfernt (mehr als 10 mm oder 20 Grad).
`ERR_STARTMOVE`	Der Roboter befindet sich bei Ausführung der Instruktion `StartMoveRetry` im Haltezustand.
`ERR_PROGSTOP`	Der Roboter befindet sich bei Ausführung der Instruktion `StartMoveRetry` im Zustand Programmstopp.
`ERR_ALRDY_MOVING`	Der Roboter bewegt sich bereits bei der Abarbeitung der Instruktion `StartMoveRetry`.

Einschränkungen

Kann nur in einer `ERROR`-Behandlungsroutine in einer Bewegungstask ausgeführt werden.

In einem MultiMove-System, das koordinierte synchronisierte Bewegungen ausführt, müssen folgende Programmierregeln in der `ERROR`-Behandlungsroutine beachtet werden:

- `StartMoveRetry` muss in allen koordinierten Programmtasks verwendet werden.
- Wenn in einer `ERROR`-Behandlungsroutine Bewegungen ausgeführt werden sollen, muss in allen koordinierten Programmtasks `StorePath ... RestoPath` verwendet werden.
- Das Programm muss den Roboter vor der Abarbeitung von `RestoPath` zurück in die Fehlerposition versetzen, wenn der Roboter auf der `StorePath`-Ebene bewegt wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.265 StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung

RobotWare - OS

Fortsetzung

Syntax

```
StartMoveRetry ';'
```

Weitere Informationen

Information über	Siehe
Stoppen von Bewegungen	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Fortfahren mit einer Bewegung	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806*
Wiederaufnahme der Abarbeitung nach einem Fehler	*RETRY - Wiederaufnahme der Abarbeitung nach einem Fehler auf Seite 622*
Speichern/Wiederherstellen der Bahn	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.266 STCalib - Kalibrieren eines Servowerkzeugs

Verwendung

`STCalib` wird für die Kalibrierung des Abstands zwischen Elektroden verwendet. Dies ist nach dem Elektroden- oder Werkzeugwechsel erforderlich und wird nach dem Putzen der Elektroden oder längerem Werkzeugeinsatz empfohlen.

Hinweis: Bei der Kalibrierung führt das Werkzeug zwei Schließ-/Öffnungsbewegungen durch. Bei der ersten Schließbewegung wird die Elektrodenkontaktposition ermittelt.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `STCalib`:

Beispiel 1:

```
VAR num curr_tip_wear;
VAR num retval;
CONST num max_adjustment := 20;

STCalib gun1 \ToolChg;
```

Kalibrieren einer Servozange nach einem Werkzeugwechsel Warten auf die Beendigung der Zangenkalibrierung, bevor die nächste RAPID-Instruktion ausgeführt wird.

Beispiel 2:

```
STCalib gun1 \ToolChg \Conc;
```

Kalibrieren einer Servozange nach einem Werkzeugwechsel Fortfahren mit der nächsten RAPID-Instruktion, ohne auf die Beendigung der Zangenkalibrierung zu warten.

Beispiel 3

```
STCalib gun1 \TipChg;
```

Kalibrieren einer Servozange nach einem Elektrodenwechsel.

Beispiel 4:

```
STCalib gun1 \TipWear \RetTipWear := curr_tip_wear;
```

Kalibrieren einer Servozange nach Elektrodenverschleiß Speichern des Elektrodenverschleißes in der Variablen `curr_tip_wear`.

Beispiel 5:

```
STCalib gun1 \TipChg \RetPosAdj:=retval;
IF retval > max_adjustment THEN
TPWrite "The tips are lost!";
...
```

Kalibrieren einer Servozange nach einem Elektrodenwechsel. Überprüfung, ob die Elektroden fehlen

Beispiel 6:

```
STCalib gun1 \TipChg \PrePos:=10;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Kalibrieren einer Servozange nach einem Elektrodenwechsel. Schnelle Bewegung der Servozange in die Position `10` mm und dann Suche nach der Kontaktposition bei geringerer Geschwindigkeit.

Beispiel 7:

Beispiel für eine unzulässige Kombination:

```
STCalib gun1 \TipWear \RetTipWear := curr_tip_wear \Conc;
```

Eine Kalibrierung des Elektrodenverschleißes wird durchgeführt. Fortfahren mit der nächsten RAPID-Instruktion, ohne auf die Beendigung der Zangenkalibrierung zu warten. Der Parameter `curr_tip_wear` enthält in diesem Fall **keinen** gültigen Wert, da de Schalter `\Conc` verwendet wird (die nächste RAPID-Instruktion wird ausgeführt, bevor die Kalibrierung abgeschlossen ist).

Argumente

```
STCalib ToolName [\ToolChg] | [\TipChg] | [\TipWear] [\RetTipWear]
     [\RetPosAdj] [\PrePos] [\Conc]
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`[\ToolChg]`

Datentyp: `switch`

Kalibrierung nach einem Werkzeugwechsel

`[\TipChg]`

Datentyp: `switch`

Kalibrierung nach einem Spitzenwechsel

`[\TipWear]`

Datentyp: `switch`

Kalibrierung nach Elektrodenverschleiß

`[\RetTipWear]`

Datentyp: `num`

Erreichter Elektrodenverschleiß [mm]

`[\RetPosAdj]`

Datentyp: `num`

Positionskorrektur seit letzter Kalibrierung [mm]

`[\PrePos]`

Datentyp: `num`

Die Position, zu der mit hoher Geschwindigkeit gefahren wird, bevor die Suche nach der Kontaktposition bei geringerer Geschwindigkeit [mm] begonnen wird

`[\Conc]`

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.266 STCalib - Kalibrieren eines Servowerkzeugs
Servo Tool Control
Fortsetzung

Nachfolgende Instruktionen werden abgearbeitet, während sich die Zange bewegt. Das Argument kann verwendet werden, um die Zyklusdauer zu verkürzen. Das ist beispielsweise dann nützlich, wenn zwei Zangen gleichzeitig gesteuert werden.

Programmabarbeitung

Kalibrierungsmodi

Wenn die mechanische Einheit vorhanden ist, wird das Servowerkzeug zur Kalibrierung aufgefordert. Die Kalibrierung wird anhand der Schalter ausgeführt (siehe unten). Wenn der Parameter `RetTipWear` verwendet wird, findet eine Aktualisierung des Elektrodenverschleißes statt.

Kalibrierung nach Werkzeugwechsel:

In einer Sequenz wird das Werkzeug langsam bis zum Zusammentreffen der Elektroden geschlossen, schnell geöffnet, schnell bei geringer Kraft geschlossen und wieder geöffnet. Der Elektrodenverschleiß bleibt unverändert.

Kalibrierung nach Elektrodenwechsel:

In einer Sequenz wird das Werkzeug langsam bis zum Zusammentreffen der Elektroden geschlossen, schnell geöffnet, schnell bei geringer Kraft geschlossen und wieder geöffnet. Der Elektrodenverschleiß wird zurückgesetzt.

Kalibrierung nach Elektrodenverschleiß:

In einer Sequenz wird das Werkzeug schnell bis zum Zusammentreffen der Elektroden geschlossen, schnell geöffnet, schnell bei geringer Kraft geschlossen und wieder geöffnet. Der Elektrodenverschleiß wird aktualisiert.

Hinweis: Wenn der Schalter `Conc` verwendet wird, wird die Instruktion nach dem Start als bereit angesehen. Daher ist der Rückgabewert `RetTipWear` nicht verfügbar. In diesem Fall wird `RetTipWear` über die Funktion `STIsOpen` zurückgegeben. Weitere Informationen finden Sie unter RobotWare - OS-Funktionen - `STIsOpen`.

Positionskorrektur

Das optionale Argument `RetPosAdj` kann verwendet werden, um z. B. festzustellen, ob die Elektroden nach einem Elektrodenwechsel noch vorhanden sind. Der Parameter enthält den Wert der Positionskorrektur nach der letzten Kalibrierung. Es kann sich um einen negativen oder positiven Wert handeln.

Verwendung einer Vorposition

Um die Kalibrierung zu beschleunigen ist es möglich, eine Vorposition zu definieren. Zu Beginn der Kalibrierung wird der Zangenarm schnell zur Vorposition bewegt, dort angehalten und dann langsam*) vorwärts bewegt, um die Elektrodenkontaktposition zu ermitteln. Wählen Sie die Vorposition sorgfältig aus. Es ist wichtig, dass die Spitzen erst *nach* Erreichen der Vorposition miteinander in Kontakt kommen. Andernfalls wird die Kalibrierung ungenau und es können Fehler in der Bewegungsüberwachung auftreten. Eine Vorposition wird ignoriert, wenn sie größer als die aktuelle Zangenposition ist (um die Kalibrierung nicht zu verlangsamen).

*) Die zweite Bewegung wird ebenfalls schnell ausgeführt, wenn die Option `\TipWear` verwendet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_SGUN	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.
ERR_SGUN_ESTOP	Not-Halt bei der Servowerkzeugbewegung. Die Instruktion wurde von einer Hintergrundtask aufgerufen und es tritt ein Not-Halt ein, daher wird die Instruktion beendet. Beachten Sie, dass, wenn die Instruktion von der Haupttask aufgerufen wird, der Programmzeiger aufgrund der Instruktion angehalten, und die Instruktion wird beim Programmneustart erneut von Beginn an abgearbeitet.
ERR_SGUN_MOTOFF	Die Instruktion wird von einem Hintergrundtask aufgerufen und das System befindet sich im Zustand „Motoren aus“.
ERR_SGUN_NEGVAL	Das Argument PrePos ist mit einem Wert angegeben, der kleiner als Null ist.
ERR_SGUN_NOTACT	Die mechanische Einheit des Servowerkzeugs ist nicht aktiviert. Verwenden Sie zur Aktivierung des Servowerkzeugs die Instruktion ActUnit.
ERR_SGUN_NOTINIT	Die Position des Servowerkzeugs wurde nicht initialisiert. Die Position des Servowerkzeugs muss bei der erstmaligen Montage der Zange oder nach der Durchführung einer Feinkalibrierung initialisiert werden. Verwenden Sie die Serviceroutine ManServiceCalib oder führen Sie eine Kalibrierung nach Elektrodenwechsel durch. Der Elektrodenverschleiß wird zurückgesetzt.
ERR_SGUN_NOTOPEN	Die Zange ist nicht geöffnet, wenn STCalib aufgerufen wird.
ERR_SGUN_NOTSYNC	Die Elektroden des Servowerkzeugs sind nicht synchronisiert. Die Elektroden des Servowerkzeugs müssen synchronisiert werden, wenn der Umdrehungszähler verloren und/oder aktualisiert wurde. Prozessdaten wie Elektrodenverschleiß bleiben erhalten.

Syntax

```
STCalib
  [ 'ToolName' :=' ] < expression (IN) of string > ','
  [ '\'ToolChg] | ['\'TipChg] | [ '\'TipWear]
  [' \'RetTipWear' :=' < variable or persistent(INOUT) of num >
       ]';'
  [ '\'RetPosAdj' :=' < variable or persistent(INOUT) of num > ]';'
  [ '\'PrePos' :=' < expression (IN) of num > ]'
  [ '\'Conc' ];'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.266 STCalib - Kalibrieren eines Servowerkzeugs

Servo Tool Control

Fortsetzung

Weitere Informationen

Information über	Siehe
Öffnen eines Servowerkzeugs	*STOpen - Öffnen eines Servowerkzeugs auf Seite 836*
Schließen eines Servowerkzeugs	*STClose - Schließen eines Servowerkzeugs auf Seite 817*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.267 STClose - Schließen eines Servowerkzeugs

Verwendung

`STClose` wird zum Schließen des Servowerkzeugs verwendet.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `STClose`:

Beispiel 1:

```
VAR num curr_thickness1;
VAR num curr_thickness2;

STClose gun1, 1000, 5;
```

Schließen der Servozange bei einer Elektrodenkraft von `1000` N und einer Blechdicke von `5` mm Warten auf das Schließen der Zange, bevor die nächste RAPID-Instruktion ausgeführt wird.

Beispiel 2:

```
STClose gun1, 2000, 3\RetThickness:=curr_thickness;
```

Schließen der Servozange bei einer Elektrodenkraft von `2000` N und einer Blechdicke von `3` mm. Abrufen der gemessenen Dicke in der Variablen `curr_thickness`.

Beispiel 3

Simultanmodus:

```
STClose gun1, 1000, 5 \Conc;
STClose gun2, 2000, 3 \Conc;
```

Schließen der Servozange `gun1` mit der Elektrodenkraft `1000` N und der Blechdicke `5` mm. Fortsetzen der Abarbeitung des RAPID-Programms, ohne auf das Schließen von `gun1` zu warten, und Schließen der Servozange `gun2` mit der Elektrodenkraft `2000` N und der Bleckdicke `3` mm. Fortsetzen der Abarbeitung des RAPID-Programms, ohne auf das Schließen von `gun2` zu warten.

Beispiel 4:

```
IF STIsClosed (gun1)\RetThickness:=curr_thickness1 THEN
  IF curr_thickness1 < 0.2 Set weld_start1;
ENDIF
IF STIsClosed (gun2)\RetThickness:=curr_thickness2 THEN
  IF curr_thickness2 < 0.2 Set weld_start2;
ENDIF
```

Abrufen der gemessenen Dicke in der Variable `curr_thickness1` und `curr_thickness2` mit der Funktion `STIsClosed`.

Beispiel 5:

Beispiel für eine unzulässige Kombination:

```
STClose gun1, 2000, 3\RetThickness:=curr_thickness \Conc;
```

Schließen der Servozange und Fortsetzen der Abarbeitung des RAPID-Programms. Der Parameter `curr_thickness` enthält in diesem Fall keinen gültigen Wert, da

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.267 STClose - Schließen eines Servowerkzeugs
Servo Tool Control
Fortsetzung

der Schalter \Conc verwendet wird (die nächste RAPID-Instruktion wird ausgeführt, bevor die Kalibrierung abgeschlossen ist).

Argumente

```
STClose ToolName TipForce Thickness [\RetThickness][\Conc]
```

ToolName

Datentyp: string

Name der mechanischen Einheit

TipForce

Datentyp: num

Die gewünschte Elektrodenkraft [N].

Thickness

Datentyp: num

Erwartete Kontaktposition für das Servowerkzeug [mm].

[\RetThickness]

Datentyp: num

Die erzielte Dicke [mm]; sie erhält nur dann einen Wert, wenn der Schalter \Conc nicht verwendet wird.

[\Conc]

Datentyp: switch

Nachfolgende Instruktionen werden abgearbeitet, während sich die Zange bewegt. Das Argument kann verwendet werden, um die Zyklusdauer zu verkürzen. Das ist beispielsweise dann nützlich, wenn zwei Zangen gleichzeitig gesteuert werden.

Programmabarbeitung

Wenn die mechanische Einheit existiert, erhält das Servowerkzeug die Anweisung, sich bis auf die erwartete Dicke und Kraft zu schließen.

Beim Schließen bewegt sich der Werkzeugarm zur erwarteten Kontaktposition (Blechdicke). In dieser Position wird die Bewegung angehalten und es erfolgt ein Wechsel vom Positionssteuerungsmodus in den Kraftsteuerungsmodus.

Der Werkzeugarm wird bei maximaler Geschwindigkeit und Beschleunigung bewegt, wie in den Systemparametern für entsprechende externe Achsen definiert. Bei den anderen Achsenbewegungen wird die Geschwindigkeit auf Einrichtbetrieb reduziert.

Sobald die gewünschte Elektrodenkraft erreicht wurde, ist die Instruktion bereit und die erreichte Dicke wird zurückgegeben, wenn das optionale Argument RetThickness angegeben wurde.

Hinweis: Wenn der Schalter Conc verwendet wird, wird die Instruktion nach dem Start als bereit angesehen. Daher ist der Rückgabewert RetThickness nicht verfügbar. In diesem Fall wird RetThickness über die Funktion STIsClosed zurückgegeben. Weitere Informationen finden Sie in den Funktionen des RobotWare-Betriebssystems - STIsClosed.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das Werkzeug kann bei einer programmierten Roboterbewegung geschlossen werden, solange der Werkzeugarm bei der Roboterbewegung nicht bewegt werden muss.

Weitere Informationen erhalten Sie in der Dokumentation zur Bewegungssteuerung des Servowerkzeugs.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_SGUN	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.
ERR_SGUN_ESTOP	Not-Halt bei der Servowerkzeugbewegung. Die Instruktion wurde von einer Hintergrundtask aufgerufen und es tritt ein Not-Halt ein, daher wird die Instruktion beendet. Beachten Sie, dass, wenn die Instruktion von der Haupttask aufgerufen wird, der Programmzeiger aufgrund der Instruktion angehalten, und die Instruktion wird beim Programmneustart erneut von Beginn an abgearbeitet.
ERR_SGUN_MOTOFF	Die Instruktion wird von einem Hintergrundtask aufgerufen und das System befindet sich im Zustand „Motoren aus“.
ERR_SGUN_NEGVAL	Das Argument PrePos ist mit einem Wert angegeben, der kleiner als Null ist.
ERR_SGUN_NOTACT	Die mechanische Einheit des Servowerkzeugs ist nicht aktiviert. Verwenden Sie zur Aktivierung des Servowerkzeugs die Instruktion ActUnit.
ERR_SGUN_NOTINIT	Die Position des Servowerkzeugs wurde nicht initialisiert. Die Position des Servowerkzeugs muss bei der erstmaligen Montage der Zange oder nach der Durchführung einer Feinkalibrierung initialisiert werden. Verwenden Sie die Serviceroutine ManServiceCalib oder führen Sie eine Kalibrierung nach Elektrodenwechsel durch. Der Elektrodenverschleiß wird zurückgesetzt.
ERR_SGUN_NOTOPEN	Die Zange ist nicht geöffnet, wenn STClose aufgerufen wird.
ERR_SGUN_NOTSYNC	Die Elektroden des Servowerkzeugs sind nicht synchronisiert. Die Elektroden des Servowerkzeugs müssen synchronisiert werden, wenn der Umdrehungszähler verloren und/oder aktualisiert wurde. Prozessdaten wie Elektrodenverschleiß bleiben erhalten.

Syntax

```
STClose
  [ 'ToolName ':=' ] < expression (IN) of string > ','
  [ 'Tipforce' :=' ] < expression (IN) of num > ','
  [ 'Thickness' :='] < expression (IN) of num > ]
  ['\' 'RetThickness' :=' < variable or persistent (INOUT) of num
       > ]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.267 STClose - Schließen eines Servowerkzeugs

Servo Tool Control

Fortsetzung

```
['\'Conc]
```

Weitere Informationen

Information über	Siehe
Öffnen eines Servowerkzeugs	*STOpen - Öffnen eines Servowerkzeugs auf Seite 836*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.268 StepBwdPath - Rückwärtsbewegung um einen Schritt auf der Bahn

Verwendung

`StepBwdPath` wird verwendet, um den TCP aus einer `RESTART`-Ereignisroutine rückwärts auf der Roboterbahn zu bewegen.

Nur der Anwender kann ein Neustartprozess-Flag setzen, daher wird `StepBwdPath` in der `RESTART`-Ereignisroutine nur bei einem Prozessneustart und nicht bei allen Programmneustarts durchgeführt.

Diese Instruktion kann nur in der Haupttask `T_ROB1` bzw. in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `StepBwdPath`:

Beispiel 1:

```
StepBwdPath 30, 1;
StepBwdPath 30, 1;
```

Die erste Instruktion bedeutet eine Rückwärtsbewegung um 30 mm. Die zweite Instruktion bedeutet eine weitere Rückwärtsbewegung um 30 mm.

Argumente

```
StepBwdPath StepLength StepTime
```

`StepLength`

Datentyp: `num`

Gibt den Weg in mm an, der in diesem Schritt rückwärts ausgeführt werden soll. Dieses Argument muss einen positiven Wert aufweisen.

`StepTime`

Datentyp: `num`

Dieses Argument ist veraltet. Setzen Sie es auf 1.

Programmabarbeitung

Der Roboter bewegt sich auf seiner Bahn um die angegebene Entfernung zurück. Die Bahn wird wie vor dem Stopp auf dieselbe Weise rückwärts ausgeführt. Bei einem schnellen Stopp oder Not-Halt wird nach der Wiederanfahrphase die `RESTART`-Ereignisroutine aufgerufen. Der Roboter befindet sich also bereits wieder auf seiner Bahn, wenn diese Instruktion abgearbeitet wird.

Die tatsächliche Geschwindigkeit dieser Bewegung ist die programmierte Geschwindigkeit, begrenzt auf 250 mm/s.

Die folgenden Eigenschaften sind in einem MultiMove-System - koordinierte Synchronbewegungen gültig:

- Alle beteiligten mechanischen Einheiten werden gleichzeitig und koordiniert zurück bewegt.
- Jedes abgearbeitete `StepBwdPath` in jeder beteiligten Programmtask führt zu einem neuen Rückwärtsbewegungsschritt (ohne erforderliches `StartMove`).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.268 StepBwdPath - Rückwärtsbewegung um einen Schritt auf der Bahn
RobotWare - OS
Fortsetzung

- Um die unterbrochenen Prozessbewegungen neu zu starten und fortzusetzen, muss die Instruktion `StartMove` in allen beteiligten Programmtasks abgearbeitet werden.

Fehlerbehandlung

Bei dem Versuch, sich über diese Grenzen hinauszubewegen, wird die Fehlerbehandlungsroutine aufgerufen und `ERRNO` auf `ERR_BWDLIMIT` gesetzt.

Einschränkungen

Nach dem Programmstopp kann die Bahn mit folgenden Einschränkungen rückwärts vollzogen werden:

- Die `StepBwdPath`-Bewegungen sind auf den letzten Feinpunkt beschränkt, und es kann eine Folge von maximal fünf Bewegungen nacheinander festgelegt werden.

Syntax

```
StepBwdPath
  [ StepLength':=' ] < expression (IN) of num >','
  [ StepTime ':=' ] < expression (IN) of num >';'
```

Weitere Informationen

Information über	Siehe
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.269 STIndGun - Versetzen des Servowerkzeug in den Modus „Unabhängig“.

Verwendung

`STIndGun` (*Servo Tool independent gun*) wird verwendet, um das Servowerkzeug (z. B. Zange oder Greifer) in den Modus „Unabhängig“ zu versetzen und das Werkzeug danach an eine angegebene unabhängige Position zu bewegen. Das Werkzeug bleibt im Modus „Unabhängig“, bis die Instruktion `STIndGunReset` ausgeführt wird.

Während des unabhängigen Modus ist das Werkzeug vom Roboter getrennt. Das Werkzeug kann geschlossen, geöffnet, kalibriert oder an eine neue unabhängige Position gebracht werden, folgt jedoch nicht koordinierten Roboterbewegungen.

Der Modus „Unabhängig“ ist von Nutzen, wenn das Werkzeug eine Aufgabe ausführt, die von der Aufgabe des Roboters unabhängig ist, wie beispielsweise die Elektrodenaufbereitung einer stationären Zange.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `STIndGun`:

Beispiel 1:

Diese Prozedur kann über eine Hintergrundtask ausgeführt werden, während der Roboter in der Haupttask beispielsweise die Ausführung von Bewegungsinstruktionen fortsetzen kann.

```
PROC tipdress()

  ! Note that the gun will move to current robtarget position, if
  ! already in independent mode.
  STIndGunReset gun1;
  ...
  STIndGun gun1, 30;
  StClose gun1, 1000, 5;
  WaitTime 10;
  STOpen gun1;
  ...
  STIndGunReset gun1;

ENDPROC
```

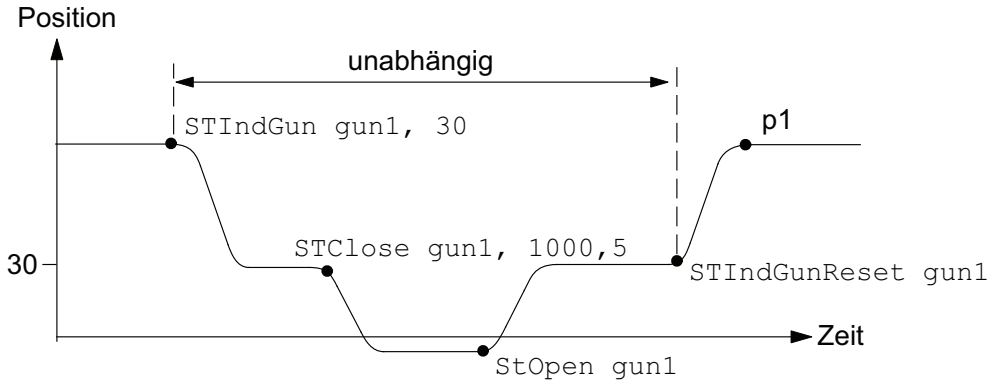
Der Modus „Unabhängig“ wird aktiviert und die Zange wird an die unabhängige Position bewegt (`30` mm). Solange sich die Zange in diesem Modus befindet, werden die Instruktionen `StClose`, `WaitTime` und `STOpen` abgearbeitet, ohne dass die Roboterbewegung beeinträchtigt wird. Die Instruktion `StIndGunReset`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.269 STIndGun - Versetzen des Servowerkzeug in den Modus „Unabhängig“.
Servo Tool Control
Fortsetzung

setzt die Zange aus dem Modus „Unabhängig“ zurück und bewegt sie an die aktuelle Roboterposition.



xx0500002342

Die Position p1 ist abhängig von der in der gerade vom Roboter ausgeführten Bewegung angegebenen Position der Zange.

Argumente

```
STIndGun ToolName GunPos
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`GunPos`

Datentyp: `num`

Die Position (Anschlag) der Servozange in mm.

Syntax

```
STIndGun
  [ ToolName ':=' ] < expression (IN) of string > ','
  [ GunPos ':=' < expression (IN) of num > ]';'
```


© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.270 STIndGun - Zurücksetzen des Werkzeugs aus dem Modus „Unabhängig“.

Verwendung

`STIndGunReset` (*Servo Tool independent gun reset*) wird verwendet, um das Servowerkzeug aus dem Modus "Unabhängig" zurückzusetzen und an die aktuelle Roboterposition zu bringen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `STIndGunReset`:

```
STIndGunReset gun1;
```

Argumente

```
STIndGunReset ToolName
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

Programmabarbeitung

Die Instruktion setzt das Werkzeug aus dem Modus „Unabhängig“ zurück und bringt das Werkzeug an die aktuelle Roboterposition. Während dieser Bewegung muss die koordinierte Geschwindigkeit der Zange 0 sein, da sonst ein Fehler auftritt. Die koordinierte Geschwindigkeit ist dann null, wenn der Roboter stillsteht oder die aktuelle Roboterbewegung eine „Nullbewegung“ der Zange beinhaltet

Syntax

```
STIndGunReset
  [ToolName ':=']<expression (IN) of string>';'
```


© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.271 SToolRotCalib - Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug

Verwendung

`SToolRotCalib` *(Stationary Tool Rotation Calibration)* wird zum Kalibrieren des TCP und der Drehung eines stationären Werkzeugs verwendet.

Die Position des Roboters und seine Bewegungen beziehen sich stets auf sein Werkzeug-Koordinatensystem, d. h. den TCP und die Werkzeugorientierung. Für größte Präzision ist es wichtig, das Werkzeug-Koordinatensystem so korrekt wie möglich zu definieren.

Die Kalibrierung kann auch manuell mit dem FlexPendant durchgeführt werden, siehe *Bedienungsanleitung - IRC5 mit FlexPendant*.

Beschreibung

Um den TCP und die Rotation eines stationären Werkzeugs zu definieren, muss ein bewegliches Zeigegerät am Endeffektor des Roboters montiert sein.

Vor Verwendung der Instruktion `SToolRotCalib` müssen einige Voraussetzungen erfüllt sein:

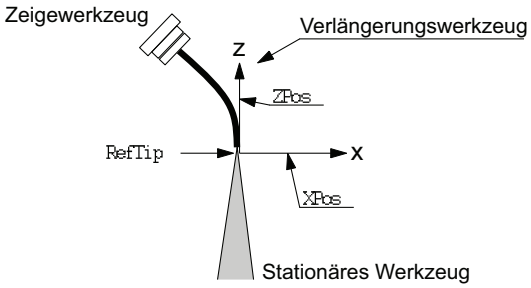
- Das zu kalibrierende stationäre Werkzeug muss am Roboter feststehend montiert und mit der korrekten Komponente robhold (`FALSE`) definiert sein.
- Das Zeigewerkzeug (`robhold TRUE`) muss mit den korrekten TCP-Werten definiert und kalibriert sein.
- Wenn der Roboter mit Absolutgenauigkeit verwendet wird, müssen Lasten und Schwerpunkt des Zeigewerkzeugs bereits definiert sein. Mit `LoadIdentify` kann die Last definiert werden.
- Das Zeigewerkzeug `wobj0` sowie `PDispOff` müssen aktiviert sein, bevor der Roboter bewegt wird.
- Bewegen Sie den TCP des Zeigewerkzeugs so nahe wie möglich an den TCP des stationären Werkzeugs (Ursprung des Werkzeug-Koordinatensystems) und definieren Sie ein `robtarget` für den Referenzpunkt `RefTip`.
- Bewegen Sie den Roboter ohne Ändern der Werkzeugorientierung so, dass der TCP des Zeigewerkzeugs auf einen Punkt an der positiven z-Achse des Werkzeug-Koordinatensystems weist, und definieren Sie ein `robtarget` für Punkt `ZPos`.
- Bewegen Sie den Roboter ohne Ändern der Werkzeugorientierung so, dass der TCP des Zeigewerkzeugs auf einen Punkt an der positiven x-Achse des Werkzeug-Koordinatensystems weist, und definieren Sie ein `robtarget` für Punkt `XPos`.

Als Hilfsmittel zum Anzeigen der positiven z- und x-Achse kann eine Art Verlängerungswerkzeug verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Definition der Roboterziele `RefTip`, `ZPos` **und** `XPos`. **Siehe folgende Abbildung.**



xx0500002343

Hinweis

Es wird empfohlen, die Positionen `RefTip`, `ZPos` **und** `XPos` **in der Instruktion** `SToolRotCalib` **nicht zu ändern.**

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SToolRotCalib`:

Beispiel 1:

```
! Created with pointing TCP pointing at the stationary tool
! coordinate system
CONST robtarget pos_tip := [...];
CONST robtarget pos_z := [...];
CONST robtarget pos_x := [...];

PERS tooldata tool1:= [ FALSE, [[0, 0, 0], [1, 0, 0 ,0]], [0, [0,
     0, 0], [1, 0, 0, 0], 0, 0, 0]];

!Instructions for creating or ModPos of pos_tip, pos_z and pos_x
MoveJ pos_tip, v10, fine, point_tool;
MoveJ pos_z, v10, fine, point_tool;
MoveJ pos_x, v10, fine, point_tool;

SToolRotCalib pos_tip, pos_z, pos_x, tool1;
```

Die Position des TCP (`tframe.trans`**) und die Werkzeugorientierung (**`tframe.rot`**) von** `tool1` **im Welt-Koordinatensystem werden berechnet und aktualisiert.**

Argumente

```
SToolRotCalib RefTip ZPos XPos Tool
```

`RefTip`

Datentyp: `robtarget`

Der Punkt, an dem der TCP des Zeigewerkzeugs auf den zu kalibrierenden TCP des stationären Werkzeugs zeigt.

`ZPos`

Datentyp: `robtarget`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.271 SToolRotCalib - Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug
RobotWare - OS
Fortsetzung

Der Verlängerungspunkt, der die positive z-Richtung definiert.

`XPos`

Datentyp: `robtarget`

Der Verlängerungspunkt, der die positive x-Richtung definiert.

`Tool`

Datentyp: `tooldata`

Die Persistente des Werkzeugs, das kalibriert werden soll.

Programmabarbeitung

Das System berechnet und aktualisiert den TCP (`tframe.trans`) und die Werkzeugorientierung (`tfame.rot`) in den angegebenen `tooldata`. Die Berechnung beruht auf den angegebenen 3 `robtarget`. Die übrigen Daten in `tooldata` werden nicht geändert.

Syntax

```
SToolRotCalib
  [RefTip ':='] <expression (IN) of robtarget>','
  [ZPos ':='] <expression (IN) of robtarget>','
  [XPos ':='] <expression (IN) of robtarget>','
  [Tool ':='] <persistent (PERS) of tooldata>';'
```

Weitere Informationen

Information über	Siehe
Kalibrierung des TCP für Werkzeuge in Bewegung	*MToolTCPCalib - Kalibrierung des TCP eines Werkzeugs in Bewegung auf Seite 489*
Kalibrierung der Rotation für Werkzeuge in Bewegung	*MToolRotCalib - Kalibrierung der Rotation eines Werkzeugs in Bewegung auf Seite 486*
Kalibrierung des TCP für ein stationäres Werkzeug	*MToolTCPCalib - Kalibrierung des TCP eines Werkzeugs in Bewegung auf Seite 489*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.272 SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs

Verwendung

`SToolTCPCalib` *(Stationary Tool TCP Calibration)*) wird zum Kalibrieren des Werkzeugarbeitspunkts (TCP) eines stationären Werkzeugs verwendet.

Die Position des Roboters und seine Bewegungen beziehen sich stets auf sein Werkzeug-Koordinatensystem, d. h. den TCP und die Werkzeugorientierung. Für größte Präzision ist es wichtig, das Werkzeug-Koordinatensystem so korrekt wie möglich zu definieren.

Die Kalibrierung kann auch manuell mit dem FlexPendant durchgeführt werden (beschrieben in *Bedienungsanleitung - IRC5 mit FlexPendant*).

Beschreibung

Um den TCP eines stationären Werkzeugs zu definieren, muss ein bewegliches Zeigegerät am Endeffektor des Roboters montiert sein.

Folgende Bedingungen müssen erfüllt sein, bevor die Instruktion `SToolTCPCalib` verwendet werden kann:

- Das zu kalibrierende stationäre Werkzeug muss am Roboter feststehend montiert und mit der korrekten Komponente `robhold` (`FALSE`) definiert sein.
- Das Zeigewerkzeug (`robhold TRUE`) muss mit den korrekten TCP-Werten definiert und kalibriert sein.
- Wenn der Roboter mit Absolutgenauigkeit verwendet wird, müssen Lasten und Schwerpunkt des Zeigewerkzeugs bereits definiert sein. Mit `LoadIdentify` kann die Last definiert werden.
- Das Zeigewerkzeug `wobj0` sowie `PDispOff` müssen aktiviert sein, bevor der Roboter bewegt wird.
- Bewegen Sie den TCP des Zeigewerkzeugs so nahe wie möglich an den TCP des stationären Werkzeugs und definieren Sie ein `robtarget` für den ersten Punkt p1.
- Definieren Sie weitere drei Positionen (p2, p3 und p4) mit jeweils anderer Orientierung.
- Es empfiehlt sich, die TCP nach verschiedenen Richtungen auszurichten, um statistisch verlässliche Resultate zu erhalten.

Fortsetzung auf nächster Seite

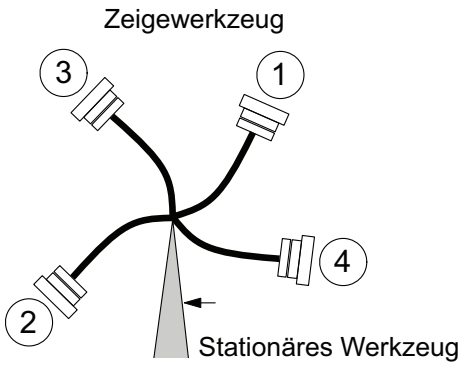
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.272 SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs

RobotWare - OS

Fortsetzung

Definition der vier robtargets p1...p4. Siehe folgende Abbildung.



xx0500002344

Hinweis

Es wird empfohlen, die Positionen `Pos1` bis `Pos4` in der Instruktion `SToolTCPCalib` nicht zu ändern.

Die Umorientierung zwischen den 4 Positionen sollte so groß wie möglich sein, so dass der Roboter unterschiedliche Konfigurationen erhält. Es empfiehlt sich außerdem, nach einer Kalibrierung die Qualität des TCP zu überprüfen. Dies kann durch Umorientierung des Werkzeugs erfolgen, um zu überprüfen, ob der TCP nicht in Bewegung ist.

Grundlegendes Beispiel

Das folgende Beispiel veranschaulicht die Instruktion `SToolTCPCalib`:

Beispiel 1:

```
! Created with pointing TCP pointing at the stationary TCP
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
CONST robtarget p4 := [...];

PERS tooldata tool1:= [ FALSE, [[0, 0, 0], [1, 0, 0 ,0]], [0,001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0]];
VAR num max_err;
VAR num mean_err;
! Instructions for creating or ModPos of p1 - p4
MoveJ p1, v10, fine, point_tool;
MoveJ p2, v10, fine, point_tool;
MoveJ p3, v10, fine, point_tool;
MoveJ p4, v10, fine, point_tool;

SToolTCPCalib p1, p2, p3, p4, tool1, max_err, mean_err;
```

Der TCP-Wert (`tframe.trans`) von `tool1` wird kalibriert und aktualisiert. `max_err` und `mean_err` geben die maximale Abweichung in mm bzw. die mittlere Abweichung in mm vom errechneten TCP an.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
SToolTCPCalib Pos1 Pos2 Pos3 Pos4 Tool MaxErr MeanErr
```

`Pos1`

Datentyp: `robtarget`

Der erste Näherungspunkt.

`Pos2`

Datentyp: `robtarget`

Der zweite Näherungspunkt.

`Pos3`

Datentyp: `robtarget`

Der dritte Näherungspunkt.

`Pos4`

Datentyp: `robtarget`

Der vierte Näherungspunkt.

`Tool`

Datentyp: `tooldata`

Die Persistente des Werkzeugs, das kalibriert werden soll.

`MaxErr`

Datentyp: `num`

Der maximale Fehler in mm für einen Näherungspunkt.

`MeanErr`

Datentyp: `num`

Der durchschnittliche Abstand der Näherungspunkte vom berechneten TCP, d. h., wie präzise der Roboter in Relation zum stationären TCP positioniert war.

Programmabarbeitung

Das System berechnet und aktualisiert den TCP-Wert im Welt-Koordinatensystem (`tframe.trans`) in den angegebenen `tooldata`. Die Berechnung beruht auf den angegebenen 4 `robtarget`. Die übrigen Daten in tooldata, wie z. B. die Werkzeugorientierung (`tframe.rot`), werden nicht geändert.

Syntax

```
SToolTCPCalib
  [ Pos1 ':=' ] < expression (IN) of robtarget > ','
  [ Pos2 ':=' ] < expression (IN) of robtarget > ','
  [ Pos3 ':=' ] < expression (IN) of robtarget > ','
  [ Pos4 ':=' ] < expression (IN) of robtarget > ','
  [ Tool ':=' ] < persistent (PERS) of tooldata > ','
  [ MaxErr ':=' ] < variable (VAR) of num > ','
  [ MeanErr ':=' ] < variable (VAR) of num > ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.272 SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Kalibrierung des TCP für Werkzeuge in Bewegung	*SToolTCPCalib - Kalibrierung des TCP eines stationären Werkzeugs auf Seite 829*
Kalibrierung der Rotation für Werkzeuge in Bewegung	*MToolRotCalib - Kalibrierung der Rotation eines Werkzeugs in Bewegung auf Seite 486*
Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug	*SToolRotCalib - Kalibrierung des TCP und der Rotation für ein stationäres Werkzeug auf Seite 826*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.273 Stop - Stoppt die Programmabarbeitung

Verwendung

`Stop` Mit wird die Programmabarbeitung angehalten. Jede zu diesem Zeitpunkt ausgeführte Bewegung wird beendet, bevor die `Stop`-Instruktion bereit ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `Stop`:

Siehe auch *Weitere Beispiele auf Seite 835*.

Beispiel 1:

```
TPWrite "The line to the host computer is broken";
Stop;
```

Die Programmabarbeitung stoppt, nachdem eine Meldung am FlexPendant ausgegeben wurde.

Argumente

```
Stop [ \NoRegain ] | [ \AllMoveTasks ]
```

`[ \NoRegain ]`

Datentyp: `switch`

Gibt für den nächsten Programmstart an, ob die betroffene mechanische Einheit wieder zur Stoppposition bewegt werden soll.

Wenn das Argument `\NoRegain` gesetzt ist, bewegen sich Roboter und externe Achsen nicht an die Stoppposition zurück (falls sie manuell von dieser wegbewegt wurden).

Wenn das Argument ausgelassen wird und der Roboter oder die externen Achsen manuell von der Stoppposition wegbewegt wurden, zeigt das FlexPendant eine Frage an. Der Anwender kann dann antworten, ob der Roboter an die Stoppposition zurückkehren soll.

`[ \AllMoveTasks ]`

Datentyp: `switch`

Gibt an, dass neben dem Programm der aktuellen Task auch die Programme in allen ausgeführten Tasks vom Typ NORMAL angehalten werden sollen.

Falls das Argument weggelassen wird, wird nur das Programm in der Task angehalten, in der die Instruktion ausgeführt wird.

Programmabarbeitung

Die Instruktion unterbricht die Programmabarbeitung, wenn die betroffenen mechanischen Einheiten in der aktuellen Bewegungstask bei der Bewegung, die gerade ausgeführt wird, eine Geschwindigkeit von null erreichen und zum Stillstand kommen. Die Programmabarbeitung kann dann ab der nächsten Instruktion neu gestartet werden.

Wenn die Instruktion ohne Schalter verwendet wird, ist nur das Programm in dieser Task betroffen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.273 Stop - Stoppt die Programmabarbeitung

RobotWare - OS

Fortsetzung

Wenn der Schalter `AllMoveTasks` in einer Task (normal, statisch oder semistatisch) verwendet wird, werden das Programm in dieser Task und alle Tasks vom Typ NORMAL angehalten. Weitere Informationen über die Deklaration von Tasks finden Sie in der Dokumentation zu Systemparametern.

Der Schalter `NoRegain` kann nur in Bewegungstasks verwendet werden, da er ausschließlich für die Bewegungsbahn von Bedeutung ist.

Wenn eine Ereignisroutine eine `Stop`-Instruktion enthält, wird die Routine angehalten und die Ausführung wie in *`Stop` auf Seite 834* beschrieben fortgesetzt.

Wenn eine Ereignisroutine in einem `MultiMove`-System eine `Stop\AllMoveTasks`-Instruktion enthält, wird der Task, der die Instruktion enthält, wie in *`Stop` auf Seite 834* beschrieben fortgesetzt. Alle anderen in einer Ereignisroutine ausgeführten Bewegungstasks werden wie in *`Stop \AllMoveTasks` auf Seite 834* beschrieben fortgesetzt (dieselbe Auswirkung wie bei normalem Programmstopp während der Abarbeitung der Ereignisroutine).

`Stop`

Ereignisroutinen	Auswirkung der Instruktion `Stop`
`POWER ON`	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl fortgesetzt.
`START`	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl fortgesetzt.
`RESTART`	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl fortgesetzt.
`STOP`	Die Ausführung wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl nicht fortgesetzt.
`QSTOP`	Die Ausführung wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl nicht fortgesetzt.
`RESET`	Die Ausführung wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl nicht fortgesetzt.

`Stop \AllMoveTasks`

Ereignisroutinen	Auswirkung der Instruktion `Stop \AllMoveTasks`
`POWER ON`	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl fortgesetzt.
`START`	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl fortgesetzt.
`RESTART`	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl fortgesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Ereignisroutinen	Auswirkung der Instruktion Stop \AllMoveTasks
STOP	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl nicht fortgesetzt.
QSTOP	Die Ausführung aller Vorgänge wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl nicht fortgesetzt.
RESET	Die Ausführung wird angehalten. Die Ausführung in der Ereignisroutine wird beim nächsten Startbefehl nicht fortgesetzt.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `Stop` sind unten aufgeführt.

Beispiel 1:

```
MoveL p1, v500, fine, tool1;
TPWrite "Jog the robot to the position for pallet corner 1";
Stop \NoRegain;
p1_read := CRobT(\Tool:=tool1 \WObj:=wobj0);
MoveL p2, v500, z50, tool1;
```

Die Programmabarbeitung stoppt mit dem Roboter an Position `p1`. Der Bediener bewegt den Roboter manuell an die Position `p1_read`. Für den nächsten Programmstart fährt der Roboter nicht an `p1` zurück, daher kann die Position `p1_read` im Programm gespeichert werden.

Syntax

```
Stop
  [ '\' NoRegain ] '|'
  [ '\' AllMoveTasks ]';'
```

Weitere Informationen

Information über	Siehe
Beenden der Programmabarbeitung	*EXIT - Beenden der Programmabarbeitung auf Seite 213*
Nur Roboterbewegungen stoppen	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Stoppen des Programms zur Fehlersuche	*Break - Abbruch der Programmabarbeitung auf Seite 52*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.274 STOpen - Öffnen eines Servowerkzeugs

Verwendung

`STOpen` wird zum Öffnen des Servowerkzeugs verwendet.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `STOpen`:

Beispiel 1:

```
STOpen gun1;
```

Öffnen des Servowerkzeugs `gun1` Warten auf das Öffnen der Zange, bevor die nächste RAPID-Instruktion ausgeführt wird.

Beispiel 2:

```
STOpen gun1 \Conc;
```

Öffnen des Servowerkzeugs `gun1` Fortfahren mit der nächsten RAPID-Instruktion, ohne auf das Öffnen der Zange zu warten.

Beispiel 3

```
STOpen "SERVOGUN"\WaitZeroSpeed;
```

Stoppen des Servowerkzeugs `SERVOGUN`, ggf. Warten, bis koordinierte Bewegungen abgeschlossen wurden, und anschließend Öffnen des Servowerkzeugs `SERVOGUN`.

Argumente

```
STOpen ToolName [\WaitZeroSpeed] [\Conc]
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`[\WaitZeroSpeed]`

Datentyp: `switch`

Stoppen des Servowerkzeugs, ggf. Warten, bis koordinierte Bewegungen abgeschlossen wurden, und anschließend Öffnen des Servowerkzeugs.

`[\Conc]`

Datentyp: `switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich die Zange bewegt. Das Argument kann verwendet werden, um die Zyklusdauer zu verkürzen. Das ist beispielsweise dann nützlich, wenn zwei Zangen gleichzeitig gesteuert werden.

Programmabarbeitung

Wenn die mechanische Einheit existiert, wird das Servowerkzeug zum Öffnen aufgefordert. Die Elektrodenkraft wird auf null verringert und der Werkzeugarm zur Position pre_close zurückbewegt.

Der Werkzeugarm wird bei maximaler Geschwindigkeit und Beschleunigung bewegt, wie in den Systemparametern für entsprechende externe Achsen definiert. Bei den anderen Achsenbewegungen wird die Geschwindigkeit auf Einrichtbetrieb reduziert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das Werkzeug kann bei einer programmierten Roboterbewegung geöffnet werden, solange der Werkzeugarm bei der Roboterbewegung nicht bewegt werden muss. Falls das Werkzeug während einer solchen Bewegung geöffnet wird, wird der Fehler `50251 Tool opening failed` angezeigt. Der Schalter `WaitZeroSpeed` kann verwendet werden, um das Risiko des Auftretens dieses Fehlers zu verringern.

Falls der Schalter \Conc verwendet wird, wird die Instruktion als bereit betrachtet, bevor das Servowerkzeug geöffnet wird. Es wird empfohlen, `STIsOpen` nach `STOpen` zu verwenden, um Probleme im Simultanmodus zu vermeiden.

Weitere Informationen erhalten Sie in der Dokumentation zur Bewegungssteuerung des Servowerkzeugs.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_SGUN`	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.
`ERR_SGUN_NOTACT`	Die mechanische Einheit des Servowerkzeugs ist nicht aktiviert. Verwenden Sie zur Aktivierung des Servowerkzeugs die Instruktion `ActUnit`.
`ERR_SGUN_NOTINIT`	Die Position des Servowerkzeugs wurde nicht initialisiert. Die Position des Servowerkzeugs muss bei der erstmaligen Montage der Zange oder nach der Durchführung einer Feinkalibrierung initialisiert werden. Verwenden Sie die Serviceroutine `ManServiceCalib` oder führen Sie eine Kalibrierung nach Elektrodenwechsel durch. Der Elektrodenverschleiß wird zurückgesetzt.
`ERR_SGUN_NOTSYNC`	Die Elektroden des Servowerkzeugs sind nicht synchronisiert. Die Elektroden des Servowerkzeugs müssen synchronisiert werden, wenn der Umdrehungszähler verloren und/oder aktualisiert wurde. Prozessdaten wie Elektrodenverschleiß bleiben erhalten.

Hinweis

Wenn die Instruktion von einer Hintergrundtask aufgerufen wurde und ein Not-Halt eintritt, wird die Instruktion ohne Fehler beendet.

Syntax

```
STOpen
[ 'ToolName ':=' ] < expression (IN) of string > ','
[ '\'WaitZeroSpeed]' ,'
['\'Conc]'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.274 STOpen - Öffnen eines Servowerkzeugs

Servo Tool Control

Fortsetzung

Weitere Informationen

Information über	Siehe
Schließen eines Servowerkzeugs	*STClose - Schließen eines Servowerkzeugs auf Seite 817*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.275 StopMove - Stoppen der Roboterbewegung

Verwendung

`StopMove` wird verwendet, um Bewegungen des Roboters und der externen Achsen und alle zugehörigen Prozesse vorübergehend zu stoppen. Wenn die Instruktion `StartMove` erteilt wird, werden die Bewegung und der Prozess wiederaufgenommen.

Diese Instruktion kann beispielsweise in einer Interrupt-Routine verwendet werden, um den Roboter temporär bei Auftreten eines Interrupts anzuhalten.

Im Basissystem kann diese Instruktion in folgenden Programmtasktypen verwendet werden:

- in der Haupttask `T_ROB1` zum Anhalten der Bewegung in dieser Task,
- in einer beliebigen anderen Task zum Anhalten der Bewegungen in der Haupttask.

In MultiMove-Systemen kann diese Instruktion in folgenden Programmtasktypen verwendet werden:

- in der Bewegungstask zum Anhalten der Bewegung in dieser Task,
- in einer Nicht- Bewegungstask zum Anhalten der Bewegung in der verbundenen Bewegungstask. Beim Stoppen der Bewegung in einer Bewegungstask aus einer koordinierten synchronisierten Taskgruppe wird zusätzlich die Bewegung in allen koordinierten Tasks gestoppt.

Hinweis

Die Sequenz `StopMove` und `StartMove` muss immer von derselben Task ausgeführt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `StopMove`:

Siehe auch *Weitere Beispiele auf Seite 840*.

Beispiel 1:

```
StopMove;
WaitDI ready_input, 1;
StartMove;
```

Die Roboterbewegung wird gestoppt, bis der Eingang `ready_input` gesetzt wird.

Argumente

```
StopMove [\Quick] [\AllMotionTasks]
```

`[\Quick]`

Datentyp: `switch`

Stoppt den Roboter so schnell wie möglich auf der Bahn.

Ohne den optionalen Parameter `\Quick` stoppt der Roboter auf der Bahn, jedoch ist der Bremsweg länger (wie beim normalen Programmstopp).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.275 StopMove - Stoppen der Roboterbewegung
RobotWare - OS
Fortsetzung

`[\AllMotionTasks]`

Datentyp: `switch`

Stoppen der Bewegung in allen mechanischen Einheiten im System. Der Schalter `[\AllMotionTasks]` kann nur aus einer Programmtask verwendet werden, die keine Bewegungstask ist.

Programmabarbeitung

Die Bewegungen des Roboters und der externen Achsen stoppen, ohne dass die Bremsen betätigt werden. Alle mit der aktuellen Bewegung verbundenen Prozesse werden gleichzeitig mit der Bewegung gestoppt.

Die Programmabarbeitung fährt fort, nachdem auf das Stoppen des Roboters und der externen Achsen (Stillstand) gewartet wurde.

Mit dem Schalter `\AllMotionTasks` (nur zulässig in Tasks, die keine Bewegungstasks sind) werden die Bewegungen für alle mechanischen Einheiten im System gestoppt.

In einem Basissystem ohne den Schalter `\AllMotionTasks` werden die Bewegungen der folgenden mechanischen Einheiten gestoppt:

- immer die der mechanischen Einheiten der Haupttask, unabhängig davon, welche Task die Instruktion `StopMove` ausführt.

In einem MultiMove-System ohne den Schalter `\AllMotionTasks` werden die Bewegungen der folgenden mechanischen Einheiten gestoppt:

- die der mechanischen Einheiten in der Bewegungstask, die die Instruktion `StopMove` ausführen.
- die mechanischen Einheiten in der Bewegungstask, die mit der Nicht-Bewegungstask verbunden sind, die `StopMove` ausführt. Beim Stoppen der mechanischen Einheiten in einer der verbundenen Bewegungstasks aus einer koordinierten synchronisierten Taskgruppe werden zusätzlich alle mechanischen Einheiten in den koordinierten Tasks gestoppt.

Der `StopMove`-Status in der Bewegungstask, der in der Bewegungstask selbst generiert wird, wird automatisch gesetzt, wenn die Task am Anfang gestartet wird.

Der `StopMove`-Status in der verbundenen Bewegungstask, der in derselben Nicht-Bewegungstask generiert wird, wird automatisch zurückgesetzt:

- wenn es sich um eine normale Nicht-Bewegungstask handelt, beim Starten der Task vom Anfang
- wenn es sich um eine Nicht-Bewegungstask vom Typ semi-static handelt, während des Neustarts nach einem Stromausfall beim Starten der Task vom Anfang
- wenn es sich um eine Nicht-Bewegungstask vom Typ static handelt, während des installation -Starts beim Starten der Task vom Anfang

Weitere Beispiele

Weitere Beispiele für die Instruktion `StopMove` sind unten aufgeführt.

Beispiel 1:

```
VAR intnum intno1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
...
PROC main()
  ...
  CONNECT intno1 WITH go_to_home_pos;
  ISignalDI di1,1,intno1;
  ...

TRAP go_to_home_pos
  VAR robtarget p10;
  StopMove;
  StorePath;
  p10:=CRobT(\Tool:=tool1 \WObj:=wobj0);
  MoveL home,v500,fine,tool1;
  WaitDI di1,0;
  Move L p10,v500,fine,tool1;
  RestoPath;
  StartMove;
ENDTRAP
```

Wenn der Eingang `di1` auf 1 gesetzt ist, wird ein Interrupt aktiviert, der die Interrupt-Routine `go_to_home_pos` aufruft. Die aktuelle Bewegung wird gestoppt und der Roboter bewegt sich stattdessen an die Position `home`. Wenn `di1` auf 0 gesetzt wird, kehrt der Roboter an die Position zurück, an der der Interrupt aufgetreten ist, und bewegt sich weiter auf der programmierten Bahn.

Beispiel 2:

```
VAR intnum intno1;
...
PROC main()
  ...
  CONNECT intno1 WITH go_to_home_pos;
  ISignalDI di1,1,intno1;
  ...

TRAP go_to_home_pos ()
  VAR robtarget p10;
  StorePath;
  p10:=CRobT(\Tool:=tool1 \WObj:=wobj0);
  MoveL home,v500,fine,tool1;
  WaitDI di1,0;
  MoveL p10,v500,fine,tool1;
  RestoPath;
  StartMove;
ENDTRAP
```

Ähnlich dem vorherigen Beispiel, jedoch bewegt sich der Roboter erst dann an die Position „home“, wenn die aktuelle Bewegungsinstruktion abgearbeitet ist.

Einschränkungen

Um die `StopMove` - `StartMove`-Sequenz für eine Bewegungstask auszuführen, ist nur eine von mehreren Nicht-Bewegungstasks gleichzeitig zulässig.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.275 StopMove - Stoppen der Roboterbewegung

RobotWare - OS

Fortsetzung

Syntax

```
StopMove
  ['\'Quick]
  ['\'AllMotionTasks]';'
```

Weitere Informationen

Information über	Siehe
Fortfahren mit einer Bewegung	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806* *StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung auf Seite 809*
Speichern/Wiederherstellen der Bahn	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.276 StopMoveReset - Zurücksetzen des Systemstatus für das Stoppen der Bewegung

Verwendung

`StopMoveReset` wird verwendet, um den Systemstatus für das Stoppen der Bewegung zurückzusetzen, ohne eine Bewegung zu starten.

Asynchron ausgelöste Fehler wie `ERR_PATH_STOP` oder bestimmte Prozessfehler während Bewegungen können mit der `ERROR`-Behandlungsroutine behoben werden. Wenn ein solcher Fehler auftritt, wird die Bewegung sofort gestoppt und das Systemflag für das Stoppen der Bewegung wird für die tatsächliche Programmtask gesetzt. Das bedeutet, dass die Bewegung nicht neu gestartet wird, falls ein Programmstart durchgeführt wird, während sich der Programmzeiger in der `ERROR`-Behandlungsroutine befindet.

Der Neustart der Bewegung nach einem derartigen Bewegungsfehler wird durch eine der folgenden Aktionen durchgeführt:

- Ausführen von `StartMove` oder `StartMoveRetry`.
- Ausführen von `StopMoveReset`, und die Bewegung wird beim nächsten Programmstart neu gestartet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `StopMoveReset`:

Beispiel 1:

```
...
ArcL p101, v100, seam1, weld1, weave1, z10, gun1;
...
ERROR
  IF ERRNO=AW_WELD_ERR OR ERRNO=ERR_PATH_STOP THEN
    ! Execute something but without any restart of the movement
    ! ProgStop - ProgStart must be allowed
    ...
    ! No idea to try to recover from this error, so let the error
    ! stop the program
    ...
    ! Reset the move stop flag, so it’s possible to manual restart
    ! the program and the movement after that the program has
    ! stopped
    StopMoveReset;
  ENDIF
ENDPROC
```

Nachdem die obige `ERROR`-Behandlungsroutine `ENDPROC` ausgeführt hat, wird die Programmabarbeitung angehalten und der Zeiger befindet sich am Anfang der Instruktion `ArcL`. Der nächste Programmstart startet das Programm und die Bewegung erneut an der Position, an der der ursprüngliche Bewegungsfehler aufgetreten ist.

Argumente

```
StopMoveReset [\AllMotionTasks]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.276 StopMoveReset - Zurücksetzen des Systemstatus für das Stoppen der Bewegung

RobotWare - OS

Fortsetzung

`[\AllMotionTasks]`

Datentyp: `switch`

Setzt den Systemstatus für das Stoppen der Bewegung aller mechanischen Einheiten im System zurück. Der Schalter `[\AllMotionTasks]` kann nur aus einer Programmtask verwendet werden, die keine Bewegungstask ist.

Programmabarbeitung

Um eine MultiMove-Anwendung im synchronisierten koordinierten Modus zurückzusetzen, muss `StopMoveReset` in allen Bewegungstasks ausgeführt werden, die koordiniert werden.

Mit dem Schalter `\AllMotionTasks` (nur zulässig in Tasks, die keine Bewegungstasks sind) werden alle mechanischen Einheiten im System zurückgesetzt.

In einem Basissystem ohne den Schalter `\AllMotionTasks` wird immer die Haupttask zurückgesetzt, unabhängig davon, von welcher Task die `StopMoveReset`-Instruktion ausgeführt wird.

Im Basissystem kann `StopMoveReset` in folgenden Programmtasktypen verwendet werden:

- Haupttask `T_ROB1`, um den Status für das Stoppen der Bewegung in dieser Task zurückzusetzen.
- beliebige andere Task, um den Status für das Stoppen der Bewegung in der Haupttask zurückzusetzen.

In einem MultiMove-System kann diese Instruktion in folgenden Programmtasktypen verwendet werden:

- Bewegungstask, um den Status für das Stoppen der Bewegung in dieser Task zurückzusetzen.
- Nicht-Bewegungstask, um den Status für das Stoppen der Bewegung in der verbundenen Bewegungstask zurückzusetzen. Wenn der Status für das Stoppen der Bewegung in einer verbundenen Bewegungstask, die zu einer koordinierten synchronisierten Taskgruppe gehört, zurückgesetzt wird, wird der Status für das Stoppen der Bewegung in allen koordinierten Tasks zurückgesetzt.

Syntax

```
StopMoveReset
  ['\'AllMotionTasks]';'
```

Weitere Informationen

Information über	Siehe
Stoppen der Bewegung	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Fortfahren mit einer Bewegung	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806* *StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung auf Seite 809*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.276 StopMoveReset - Zurücksetzen des Systemstatus für das Stoppen der Bewegung

RobotWare - OS

Fortsetzung

Information über	Siehe
Speichern/Wiederherstellen der Bahn	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.277 StorePath - Speichern der Bahn bei einem Interrupt

Verwendung

`StorePath` wird verwendet, um die aktuelle Bewegungsbahn z. B. bei Auftreten eines Fehlers oder Interrupts zu speichern. Die Fehlerbehandlungs- oder Interrupt-Routine kann dann eine neue temporäre Bewegung starten und schließlich die ursprüngliche Bewegung fortsetzen, die zu einem früheren Zeitpunkt gespeichert wurde.

Diese Instruktion kann z. B. verwendet werden, um eine Wartungsposition anzufahren oder eine Schweißzange zu reinigen, wenn ein Fehler auftritt.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `StorePath`:

Siehe auch *Weitere Beispiele auf Seite 847*.

Beispiel 1:

```
StorePath;
```

Die aktuelle Bewegungsbahn wird zur späteren Verwendung gespeichert. Das System wird in den unabhängigen Bewegungsmodus versetzt.

Beispiel 2:

```
StorePath \KeepSync;
```

Die aktuelle Bewegungsbahn wird zur späteren Verwendung gespeichert. Der Synchronbewegungsmodus wird beibehalten.

Argumente

```
StorePath [\KeepSync]
```

`[\KeepSync]`

Keep Synchronization

Datentyp: `switch`

Behält den Synchronbewegungsmodus nach `StorePath \KeepSync` bei. Der Schalter `KeepSync` kann nur verwendet werden, wenn sich das System vor dem Aufruf von `StorePath \KeepSync` im Synchronbewegungsmodus befindet.

Ohne den optionalen Parameter `\KeepSync` wird das System bei einem koordinierten synchronisierten MultiMove-System in den unabhängigen teilkoordinierten Bewegungsmodus versetzt. Nach der Abarbeitung von `StorePath` in allen beteiligten Tasks befindet sich das System weiterhin im teilkoordinierten Modus, falls weiterhin ein koordiniertes Werkobjekt verwendet wird. Anderenfalls befindet es sich um unabhängigen Modus. Im teilkoordinierten Modus wird empfohlen, immer mit einer Bewegung der mechanischen Einheit zu beginnen, die das Anwender-Koordinatensystem steuert, bevor `WaitSyncTask` in allen beteiligten Tasks abgearbeitet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Die aktuelle Bewegungsbahn des Roboters und der externen Achsen wird gespeichert. Anschließend kann in einer Interrupt- oder Fehlerbehandlungsroutine eine andere Bewegung gestartet werden. Wenn die Ursache für den Fehler bzw. Interrupt beseitigt wurde, kann die gespeicherte Bewegungsbahn neu gestartet werden.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `StorePath` sind unten aufgeführt.

Beispiel 1:

```
TRAP machine_ready
  VAR robtarget p1;
  StorePath;
  p1 := CRobT();
  MoveL p100, v100, fine, tool1;
  ...
  MoveL p1, v100, fine, tool1;
  RestoPath;
  StartMove;
ENDTRAP
```

Wenn ein Interrupt auftritt, der die Interrupt-Routine `machine_ready` startet, wird die aktuelle Bewegungsbahn des Roboters am Ende der Instruktion (ToPoint) gestoppt und gespeichert. Anschließend korrigiert der Roboter den Interrupt, z. B. durch Ersetzen eines Maschinenteils. Dann wird die normale Bewegung neu gestartet.

Einschränkungen

Nur die Bewegungsbahn wird mit der Instruktion `StorePath` gespeichert.

Wenn der Anwender Bewegungen auf der neuen Bahnebene anfordern will, muss die aktuelle Stoppposition direkt nach `StorePath` gespeichert werden. Vor `RestoPath` muss dann zuerst eine Bewegung an die gespeicherte Stoppposition auf der Bahn erfolgen.

Es kann nur jeweils eine Bewegungsbahn gespeichert werden.

Syntax

```
StorePath
  ['\'KeepSync]';'
```

Weitere Informationen

Information über	Siehe
Wiederherstellen einer Bahn	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.277 StorePath - Speichern der Bahn bei einem Interrupt

RobotWare - OS

Fortsetzung

Information über	Siehe
Weitere Beispiele	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620* *PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *SyncMoveResume - Festlegen von synchronisierten koordinierten Bewegungen auf Seite 869* *SyncMoveSuspend - Festlegen von unabhängigen teilkoordinierten Bewegungen auf Seite 871*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.278 STTune - Abstimmung des Servowerkzeugs

Verwendung

`STTune` wird zur Abstimmung/Änderung eines Servowerkzeugparameters verwendet. Der ursprüngliche Wert des Parameters wird vorübergehend geändert, was in den Systemparametern eingestellt wird. Der neue Abstimmungswert wird sofort nach Ausführung der Anweisung aktiv.

`STTune` ist in Abstimmungsverfahren sinnvoll. Ein Abstimmungsverfahren wird typischerweise verwendet, um einen optimalen Wert für einen Parameter zu finden. Ein Experiment (d. h. eine Programmausführung mit einer Servowerkzeugbewegung) wird wiederholt, wenn unterschiedliche Parameterabstimmungswerte verwendet werden.

`STTune` darf nicht beim Kalibrieren oder Schließen des Werkzeugs verwendet werden.

Beschreibung

`RampTorqRefOpen`

Abstimmung des Systemparameters `Ramp when decrease force`, der bestimmt, wie schnell die Kraft beim Öffnen der Zange ausgelöst wird. Die Einheit ist Nm/s und ein typischer Wert ist 200.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `ramp_torque_ref_opening`.

`RampTorqRefClose`

Abstimmung des Systemparameters `Ramp when increase force`, der bestimmt, wie schnell die Kraft beim Öffnen der Zange aufgebracht wird. Die Einheit ist Nm/s und ein typischer Wert ist 80.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `ramp_torque_ref_closing`.

`KV`

Abstimmung des Systemparameters `KV`, der zur Geschwindigkeitsbegrenzung verwendet wird. Die Einheit ist Nms/rad und ein typischer Wert ist 1. Weitere Informationen finden Sie in der Dokumentation der externen Achse.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `Kv`.

`SpeedLimit`

Abstimmung des Systemparameters `Speed limit`, der zur Geschwindigkeitsbegrenzung verwendet wird. Die Einheit ist rad/s (Motorgeschwindigkeit) und ein typischer Wert ist 60. Weitere Informationen finden Sie in der Dokumentation der externen Achse.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `speed_limit`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.278 STTune - Abstimmung des Servowerkzeugs
Servo Tool Control
Fortsetzung

`CollAlarmTorq`

Abstimmung des Systemparameters `Collision alarm torque`, der zur automatischen Kalibrierung neuer Elektroden verwendet wird. Die Einheit ist Nm (Motordrehmoment) und ein typischer Wert ist 1. Weitere Informationen finden Sie in der Dokumentation der externen Achse.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `alarm_torque`.

`CollContactPos`

Abstimmung des Systemparameters `Collision delta pos`, der zur automatischen Kalibrierung neuer Elektroden verwendet wird. Die Einheit ist m und ein typischer Wert ist 0,002. Weitere Informationen finden Sie in der Dokumentation der externen Achse.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `distance_to_contact_position`.

`CollisionSpeed`

Abstimmung des Systemparameters `Collision speed`, der zur automatischen Kalibrierung neuer Elektroden verwendet wird. Die Einheit ist m/s und ein typischer Wert ist 0,02. Weitere Informationen finden Sie in der Dokumentation der externen Achse.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *Force master*, Parameter `col_speed`.

`CloseTimeAdjust`

Konstante, positive oder negative Zeiteinstellung (s) des Moments, an dem die Werkzeugelektroden beim Schließen des Werkzeugs in Kontakt kommen. Kann für eine leichte Schließverzögerung verwendet werden, wenn das synchronisierte Vorschließen zum Schweißen verwendet wird.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *SG process*, Parameter `min_close_time_adjust`.

`ForceReadyDelayT`

Konstante Zeitverzögerung(en), bevor das Bereitsschaftssignal zum Schweißen gesendet wird, nachdem die programmierte Kraft erreicht wurde.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *SG process*, Parameter `pre_sync_delay_time`.

`PostSyncTime`

Erwartung der Auslösezeit (s) der nächsten Roboterbewegung nach einer Schweißung. Dieser Abstimmungstyp kann abgestimmt werden, um das Zangenöffnen mit der nächsten Roboterbewegung zu synchronisieren. Die Synchronisation kann fehlschlagen, wenn die Parameter zu hoch gesetzt sind.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *SG process*, Parameter `post_sync_time`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`CalibTime`

Die Wartezeit (s) während einer Kalibrierung vor der positionellen Korrektur der Werkzeugelektroden. Um optimale Ergebnisse zu erhalten, verwenden Sie nicht einen zu niedrigen Wert wie z. B. 0,5 s.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *SG process*, Parameter `calib_time`.

`CalibForceLow`

Die verwendete minimale Elektrodenkraft (N) während einer TipWear-Kalibrierung. Für beste Ergebnisse bei der Erkennung der Dicke wird empfohlen, die programmierte Mindestschweißkraft zu verwenden.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *SG process*, Parameter `calib_force_low`.

`CalibForceHigh`

Die verwendete maximale Elektrodenkraft (N) während einer TipWear-Kalibrierung. Für beste Ergebnisse bei der Erkennung der Dicke wird empfohlen, die programmierte Maximalschweißkraft zu verwenden.

Entsprechender Systemparameter: Parametergruppe *Motion*, Typ *SG process*, Parameter `calib_force_high`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `STTune`:

Beispiel 1:

```
STTune SEOLO_RG, 0.050, CloseTimeAdjust;
```

Der Servowerkzeugparameter `CloseTimeAdjust` wird vorübergehend auf 0,050 Sekunden gesetzt.

Argumente

```
STTune MecUnit TuneValue Type
```

`MecUnit`

Datentyp: `mecunit`

Name der mechanischen Einheit

`TuneValue`

Datentyp: `num`

Der neue Abstimmungswert.

`Type`

Datentyp: `tunegtype`

Der Parametertyp. Für die Abstimmung verfügbare Servowerkzeugparameter sind `RampTorqRefOpen`, `RampTorqRefClose`, `KV`, `SpeedLimit`, `CollAlarmTorq`, `CollContactPos`, `CollisionSpeed`, `CloseTimeAdjust`, `ForceReadyDelayT`, `PostSyncTime`, `CalibTime`, `CalibForceLow` und `CalibForceHigh`. Diese Typen sind in den Systemparametern vordefiniert und definieren die Originalwerte.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.278 STTune - Abstimmung des Servowerkzeugs

Servo Tool Control

Fortsetzung

Programmabarbeitung

Der spezifizierte Abstimmtyp und -wert wird für die spezifizierte mechanische Einheit aktiviert. Dieser Wert gilt für alle Bewegungen, bis ein neuer Wert für die aktuelle mechanische Einheit programmiert wird oder die Abstimmtypen und -werte mithilfe der Instruktion `STTuneReset` zurückgesetzt werden.

Die ursprünglichen Abstimmwerte können in den Systemparametern dauerhaft geändert werden.

Die Standardabstimmwerte für Servowerkzeuge werden automatisch eingestellt

- bei der Abarbeitung der Instruktion `STTuneReset`.
- bei einem **Neustart**.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_SGUN`	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.

Syntax

```
STTune
  [ MecUnit ':=' ] < variable (VAR) of mecunit > ','
  [ TuneValue' :=' ] < expression (IN) of num > ','
  [ 'Type ':='] < expression (IN) of tunegtype > ]';'
```

Weitere Informationen

Information über	Siehe
Wiederherstellen der Servowerkzeugparameter	*TuneReset - Zurücksetzen der Servo-Abstimmung auf Seite 1003*
Abstimmung des Servowerkzeugs	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.279 STTuneReset - Zurücksetzen der Servowerkzeugabstimmung

Verwendung

`STTuneReset` wird zum Wiederherstellen der ursprünglichen Werte von Servowerkzeugparametern verwendet, wenn sie durch die Instruktion `STTune` geändert wurden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `STTuneReset`:

Beispiel 1:

```
STTuneReset SEOLO_RG;
```

Wiederherstellen *der ursprünglichen Werte der Werkzeugparameter* für die mechanische Einheit `SEOLO_RG`.

Argumente

```
STTuneReset MecUnit
```

`MecUnit`

Datentyp: `mecunit`

Name der mechanischen Einheit

Programmabarbeitung

Die ursprünglichen Servowerkzeugparameter werden wiederhergestellt.

Dies kann auch bei einem **Neustart** erreicht werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_SGUN`	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.

Syntax

```
STTuneReset
  [ MecUnit ':=' ] < variable (VAR) of mecunit > ','
```

Weitere Informationen

Information über	Siehe
Abstimmung der Servowerkzeugparameter	*STTune - Abstimmung des Servowerkzeugs auf Seite 849*
Abstimmung der Servowerkzeugparameter	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.280 SupSyncSensorOff - Stopp der synchronisierten Sensorüberwachung

Verwendung

SupSyncSensorOff wird dazu verwendet, die Überwachung der Roboterbewegung und der synchronisierten Sensorbewegung zu stoppen.

Grundlegendes Beispiel

Grundlegende Beispiele für die Instruktion `SupSyncSensorOff` ist unten aufgeführt.

Beispiel

```
SupSyncSensorOff SSYNC1;
```

Der Sensor wird nicht mehr überwacht.

Argumente

```
SupSyncSensorOff MechUnit
```

`MechUnit`

Mechanical unit

Datentyp: `mecunit`

Name der mechanischen Einheit

Syntax

```
SupSyncSensorOff
  [ MechUnit ':=' ] < variable (VAR) of mecunit> ';'
```

Weitere Informationen

Information über	Siehe
Start der synchronisierten Sensorüberwachung	*SupSyncSensorOn - Start der synchronisierten Sensorüberwachung auf Seite 855*
Synchr. mit Sensor	*SyncToSensor - Synchr. mit Sensor auf Seite 875*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.281 SupSyncSensorOn - Start der synchronisierten Sensorüberwachung

Verwendung

SupSyncSensorOn wird dazu verwendet, die Überwachung der Roboterbewegung und der synchronisierten Sensorbewegung zu starten.

Grundlegendes Beispiel

Grundlegende Beispiele für die Instruktion `SupSyncSensorOn` ist unten aufgeführt.

Beispiel

```
SupSyncSensorOn Ssync1, 150, 100, 50
```

Die mechanische Einheit `Ssync1` wird überwacht, wenn sich der Sensor zwischen 50 und 150 befindet. Die Überwachung endet, wenn die Distanz zwischen Roboter und Sensor weniger als 100 beträgt.

Argumente

```
SupSyncSensorOn MechUnit MaxSyncSup SafetyDist MinSyncSup
    [\SafetyDelay]
```

`MechUnit`

Mechanical unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`MaxSyncSup`

Maximal Synchronized supervised position

Datentyp: `num`

Der Roboter überwacht den Sensor, bis der Sensor die maximale Kalibrierstellung überschreitet. Mit Überschreiten dieses Punkts wird die Überwachung eingestellt. Einheiten sind in mm angegeben.

`SafetyDist`

Safety distance

Datentyp: `num`

`Safetydist` ist der Grenzwert für die Differenz zwischen der erwarteten Maschinenposition und der echten Maschinenposition. Er muss negativ sein, d. h., das Modell sollte sich immer vor der echten Maschine bewegen. Bei abnehmenden Maschinenpositionen muss der Grenzwert negativ sein, entsprechend des maximalen negativen Positionsunterschieds (und der minimalen Vorwärtsentfernung). Bei zunehmenden Maschinenpositionen muss der Grenzwert positiv sein, entsprechend des minimalen positiven Positionsunterschieds (und der minimalen Vorwärtsentfernung).

Der Roboter löst einen Alarm aus, wenn die Distanz zwischen Roboter und Sensor unterhalb des Sicherheitsabstands liegt. Wird der Alarm ausgelöst, stoppt die Überwachung.

Einheiten sind in mm angegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.281 SupSyncSensorOn - Start der synchronisierten Sensorüberwachung

Machine Synchronization
Fortsetzung

`MinSyncSup`

Minimal synchronized supervised position

Datentyp: `num`

Der Roboter beginnt mit der Überwachung, wenn sich der Sensor innerhalb des Fensters befindet, das laut Definition von Position `MinSyncSup` bis Position `MaxSyncSup` reicht. Einheiten sind in mm angegeben.

`[\SafetyDelay]`

Safety delay

Datentyp: `num`

`SafetyDelay` wird dazu verwendet, die Verzögerung zwischen der programmierten Roboterposition und der sensorüberwachten Position anzupassen. Einheiten sind in Sekunden angegeben.

Einschränkungen

Wird `SupSynSensorOn` verwendet, bevor die Instruktion `WaitSensor` beendet ist, hält der Roboter an.

Syntax

```
SupSyncSensorOn
  [ MechUnit ':=' ] <variable (VAR) of mecunit> ','
  [ MaxSyncSup ':=' ] < expression (IN) of num > ','
  [ SafetyDist ':=' ] < expression (IN) of num > ','
  [ MinSyncSup ':=' ] < expression (IN) of num >
  [ \SafetyDelay ':=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Stopp der synchronisierten Sensorüberwachung	*SupSyncSensorOff - Stopp der synchronisierten Sensorüberwachung auf Seite 854*
Synchr. mit Sensor	*SyncToSensor - Synchr. mit Sensor auf Seite 875*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.282 SyncMoveOff - Beenden von koordinierten Synchronbewegungen

Verwendung

`SyncMoveOff` wird verwendet, um eine Sequenz synchronisierter und meist auch koordinierter Bewegungen zu beenden. Zunächst warten alle Programmtasks auf die Synchronisierung an einem Stopppunkt. Dann wird die Bewegungsplanung für die betroffenen Programmtasks in den unabhängigen Modus versetzt.

Die Instruktion `SyncMoveOff` kann nur in einem *MultiMove*-System mit der Option *Coordinated Robots* und nur in als `Motion Task` definierten Programmtasks verwendet werden.

WARNUNG

Für eine sichere Synchronisierungsfunktionalität muss jeder Treffpunkt (Parameter `SyncID`) über einen eindeutigen Namen verfügen. Der Name des Treffpunkts muss auch für die Programmtasks übereinstimmen, die sich dort treffen sollen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SyncMoveOff`:

Siehe auch *Weitere Beispiele auf Seite 859*.

Beispiel 1:

```
!Program example in task T_ROB1

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...

!Program example in task T_ROB2

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...
```

Die Programmtask, die als erste `SyncMoveOff` mit der Identität `sync2` erreicht, wartet, bis die anderen Tasks `SyncMoveOff` mit derselben Identität `sync2`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.282 SyncMoveOff - Beenden von koordinierten Synchronbewegungen
RW-MRS Synchronized
Fortsetzung

erreichen. Am Synchronisierungspunkt `sync2` wird die Bewegungsplanung für die betroffenen Programmtasks in den unabhängigen Modus versetzt. Anschließend wird die Abarbeitung der Task `T_ROB1` und `T_ROB2` fortgesetzt.

Argumente

```
SyncMoveOff SyncID [\TimeOut]
```

`SyncID`

Synchronization Identity

Datentyp: `syncident`

Variablen, die den Namen des Treffpunkts angeben, an dem die Synchronisierung aufgehoben wird. Der Datentyp `syncident` ist ein Datentyp ohne einen Wert (Non-Value). Er dient lediglich als Bezeichner zum Benennen des Punkts, an dem die Synchronisierung aufgehoben wird.

Die Variable muss definiert werden und in allen koordinierten Programmtasks den gleichen Namen besitzen. Es wird empfohlen, die Variable immer global in jeder Task zu definieren (`VAR syncident ...`).

`[\TimeOut]`

Datentyp: `num`

Maximale Dauer, für die an dem Punkt, an dem die Synchronisierung aufgehoben wird, auf die anderen Programmtasks gewartet wird. Timeout in Sekunden (Auflösung 0,001s).

Wenn diese Zeit abläuft, bevor die Programmtasks den Synchronisierungspunkt erreicht haben, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_SYNCMOVEOFF` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Wenn dieses Argument weggelassen wird, wartet die Programmtask für unbegrenzte Zeit.

Programmabarbeitung

Die Programmtask, die zuerst `SyncMoveOff` erreicht, wartet, bis alle anderen angegebenen Tasks `SyncMoveOff` mit derselben `SyncID`-Identität erreichen. An diesem Punkt `SyncID`, an dem die Synchronisierung aufgehoben wird, wird die Bewegungsplanung für die betreffenden Programmtasks in den unabhängigen Modus versetzt. Danach fahren die betreffenden Programmtasks mit der Abarbeitung fort.

Die Bewegungsplanung für die betreffenden Programmtasks wird in den unsynchronisierten Modus versetzt. Das bedeutet Folgendes:

- Alle RAPID-Programmtasks und alle Bewegungen aus diesen Tasks arbeiten wieder unabhängig voneinander.
- Die Bewegungsinstruktionen dürfen **nicht** mit einer ID-Nummer gekennzeichnet werden (siehe Instruktion `MoveL`).

Programmtasks können zum Testen über das FlexPendant-Taskauswahlfeld ausgeschlossen werden. Die Instruktionen `SyncMoveOn` und `SyncMoveOff` können

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

mit der geringeren Anzahl an Programmtasks, auch mit nur einer Programmtask, abgearbeitet werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SYNCMOVEOFF	Timeout von SyncMoveOff.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion SyncMoveOff sind unten aufgeführt.

Beispiel für einfache synchronisierte Bewegung

```
!Program example in task T_ROB1
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WObj:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WObj:= rob2_obj;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC

!Program example in task T_ROB2
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, obj2;
  WaitSyncTask sync1, task_list;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.282 SyncMoveOff - Beenden von koordinierten Synchronbewegungen

RW-MRS Synchronized

Fortsetzung

```
  MoveL p_fine, v1000, fine, obj2;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, obj2;
  MoveL * \ID:=20, v100, fine, obj2 ;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
```

Zuerst warten die Programmtasks `T_ROB1` **und** `T_ROB2` **bei** `WaitSyncTask` **mit der Identität** `sync1` **aufeinander. Sie sind mit einer Zonenbahn für die vorherigen Bewegungen programmiert, um die Zyklusdauer zu verkürzen.**

Dann warten die Programmtasks bei `SyncMoveOn` **mit der Identität** `sync2` **aufeinander. Sie wurden bei der Programmierung mit dem erforderlichen Stopppunkt für die vorherigen Bewegungen versehen. Danach wird die Bewegungsplanung für die betreffenden Programmtasks in den synchronisierten Modus versetzt.**

Anschließend bewegt `T_ROB2` **das** `obj2` **zum** `ID`**-Punkt** `10` **und** `20` **im Welt-Koordinatensystem, während** `T_ROB1` **den** `tcp1` **to** `ID`**-Punkt** `10` **und** `20` **auf dem sich bewegenden Objekt** `obj2` **bewegt.**

Dann warten die Programmtasks bei `SyncMoveOff` **mit der Identität** `sync3` **aufeinander. Sie wurden bei der Programmierung mit dem erforderlichen Stopppunkt für die vorherigen Bewegungen versehen. Danach wird die Bewegungsplanung für die betreffenden Programmtasks in den unabhängigen Modus versetzt.**

Beispiel mit Fehlerbehandlung

```
!Program example with use of time-out function
VAR syncident sync3;

...
SyncMoveOff sync3 \TimeOut := 60;
...
ERROR
  IF ERRNO = ERR_SYNCMOVEOFF THEN
    RETRY;
  ENDIF
```

Die Programmtask wartet darauf, dass die Instruktion `SyncMoveOff` **und weitere Programmtasks denselben Synchronisierungspunkt** `sync3` **erreichen. Nach einer Wartezeit von 60 Sekunden wird die Fehlerbehandlungsroutine mit** `ERRNO` **gleich** `ERR_SYNCMOVEOFF` **aufgerufen. Dann wird die Instruktion** `SyncMoveOff` **nochmals aufgerufen und die Wartezeit um 60 Sekunden verlängert.**

Beispiel mit teilkoordinierter und koordinierter Bewegung

```
!Example with semicoordinated and synchronized movement
!Program example in task T_ROB1
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
PERS wobjdata rob2_obj:= [FALSE,FALSE,"ROB_2",
     [[0,0,0],[1,0,0,0]],[[155.241,-51.5938,57.6297],
     [0.493981,0.506191,-0.501597,0.49815]]];
VAR syncident sync0;
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

PROC main()
  ...
  WaitSyncTask sync0, task_list;
  MoveL p1_90, v100, fine, tcp1 \WObj:= rob2_obj;
  WaitSyncTask sync1, task_list;
  SyncMoveOn sync2, task_list;
  MoveL p1_100 \ID:=10, v100, fine, tcp1 \WObj:= rob2_obj;
  SyncMoveOff sync3;
  !Wait until the movement has been finished in T_ROB2
  WaitSyncTask sync3, task_list;
  !Now a semicoordinated movement can be performed
  MoveL p1_120, v100, z10, tcp1 \WObj:= rob2_obj;
  MoveL p1_130, v100, fine, tcp1 \WObj:= rob2_obj;
  WaitSyncTask sync4, task_list;
  ...
ENDPROC

!Program example in task T_ROB2
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync0;
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

PROC main()
  ...
  MoveL p_fine, v1000, fine, tcp2;
  WaitSyncTask sync0, task_list;
  !Wait until the movement in T_ROB1 task is finished
  WaitSyncTask sync1, task_list;
  SyncMoveOn sync2, task_list;
  MoveL p2_100 \ID:=10, v100, fine, tcp2;
  SyncMoveOff sync3;
  !The path has been removed at SyncMoveOff
  !Perform a movement to wanted position for the object to
  !make the position available for other tasks
  MoveL p2_100, v100, fine, tcp2;
  WaitSyncTask sync3, task_list;
  WaitSyncTask sync4, task_list;
  MoveL p2_110, v100, z10, tcp2;
  ...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.282 SyncMoveOff - Beenden von koordinierten Synchronbewegungen
RW-MRS Synchronized
Fortsetzung

```
ENDPROC
```

Beim Wechseln von der teilkoordinierten zur synchronisierten Bewegung ist eine `WaitSyncTask` erforderlich (wenn die Identität `sync1` verwendet wird).

Beim Wechsel von der teilkoordinierten zur synchronisierten Bewegung muss die Task, die das Werkobjekt (`rob2_obj`) bewegt, dieses zur gewünschten Position bewegen. Anschliessend ist `WaitSyncTask` (Identität `sync3`) vor der teilkoordinierten Bewegung erforderlich.

Einschränkungen

Die Instruktion `SyncMoveOff` kann nur abgearbeitet werden, wenn sich alle betreffenden Roboter am Stopppunkt im Stillstand befinden.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`SyncMoveOff` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
SyncMoveOff
  [ SyncID ':=' ] < variable (VAR) of syncident>
  [ '\'TimeOut' :=' < expression (IN) of num> ] ';'
```

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Identität für Synchronisierungspunkte	*syncident - Identität für Synchronisierungspunkte auf Seite 1814*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Festlegen von unabhängigen Bewegungen	*SyncMoveUndo - Festlegen von unabhängigen Bewegungen auf Seite 873*
Testen, ob im synchronisierten Modus	*IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus auf Seite 1385*
MultiMove-System mit Option Coordinated robots	*Anwendungshandbuch - MultiMove*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.283 SyncMoveOn - Beginn von koordinierten Synchronbewegungen

Verwendung

`SyncMoveOn` wird verwendet, um eine Sequenz synchronisierter und meist auch koordinierter Bewegungen zu starten. Zunächst warten alle Programmtasks auf die Synchronisierung an einem Stopppunkt. Dann wird die Bewegungsplanung für die betroffenen Programmtasks in den synchronisierten Modus versetzt.

Die Instruktion `SyncMoveOn` kann nur in einem *MultiMove*-System mit der Option *Coordinated Robots* und nur in als `Motion Task` definierten Programmtasks verwendet werden.

WARNUNG

Für eine sichere Synchronisierungsfunktionalität muss jeder Treffpunkt (Parameter `SyncID`) über einen eindeutigen Namen verfügen. Der Name des Treffpunkts muss auch für die Programmtasks übereinstimmen, die sich dort treffen sollen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SyncMoveOn`:

Siehe auch *Weitere Beispiele auf Seite 865*.

Beispiel 1:

```
!Program example in task T_ROB1

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...

!Program example in task T_ROB2

PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;

...
SyncMoveOn sync1, task_list;
...
SyncMoveOff sync2;
...
```

Die Programmtask, die als erste `SyncMoveOn` mit der Identität `sync1` erreicht, wartet, bis die anderen Tasks `SyncMoveOn` mit derselben Identität `sync1` erreichen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.283 SyncMoveOn - Beginn von koordinierten Synchronbewegungen

RW-MRS Synchronized

Fortsetzung

Am Synchronisierungspunkt `sync1` wird die Bewegungsplanung für die betreffenden Programmtasks in den synchronisierten Modus versetzt. Anschließend wird die synchrone Abarbeitung von Task `T_ROB1` und `T_ROB2` fortgesetzt, bis sie `SyncMoveOff` mit derselben Identität `sync2` erreichen.

Argumente

```
SyncMoveOn SyncID TaskList [\TimeOut]
```

`SyncID`

Synchronization Identity

Datentyp:`syncident`

Variable, die den Namen des Synchronisierungspunkts (Treffpunkts) angibt. Der Datentyp `syncident` ist ein Datentyp ohne einen Wert, der nur als Bezeichner für den Synchronisierungspunkt fungiert.

Die Variable muss definiert werden und in allen koordinierten Programmtasks den gleichen Namen besitzen. Es wird empfohlen, die Variable immer global in jeder Task zu definieren (`VAR syncident ...`).

`TaskList`

Datentyp:`tasks`

Persistente Variable, die in einer Taskliste (einem Datenfeld) den Namen (`string`) der Programmtasks angibt, die am Synchronisierungspunkt mit dem Namen nach dem Argument `SyncID` warten sollen.

Die persistente Variable muss definiert werden und in allen koordinierten Programmtasks über den gleichen Namen sowie den gleichen Inhalt verfügen. Es wird empfohlen, die Variable immer global im System zu definieren (`PERS tasks ...`).

`[\TimeOut]`

Datentyp: `num`

Maximale Dauer, für die am Synchronisierungspunkt auf die anderen Programmtasks gewartet wird. Timeout in Sekunden (Auflösung 0,001 s).

Wenn diese Zeit abläuft, bevor die Programmtasks den Synchronisierungspunkt erreicht haben, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_SYNCMOVEON` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Wenn dieses Argument weggelassen wird, wartet die Programmtask für unbegrenzte Zeit.

Programmabarbeitung

Die Programmtask, die zuerst `SyncMoveOn` erreicht, wartet, bis alle anderen angegebenen Tasks `SyncMoveOn` mit derselben `SyncID`-Identität erreichen. Am Synchronisierungspunkt `SyncID` wird die Bewegungsplanung für die betreffenden Programmtasks in den synchronisierten Modus versetzt. Danach fahren die betreffenden Programmtasks mit der Abarbeitung fort.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Bewegungsplanung für die betreffenden Programmtasks wird in den synchronisierten Modus versetzt. Das bedeutet Folgendes:

- Jede Bewegungsinstruktion in einer beliebigen Programmtask der `TaskList` arbeitet synchron mit Bewegungsinstruktionen in anderen Programmtasks in der `TaskList`.
- Alle koordinierten Bewegungsinstruktionen werden in derselben Bewegungsplanung geplant und interpoliert.
- Alle Bewegungen beginnen und enden zum gleichen Zeitpunkt. Die Geschwindigkeit wird auf die längste Bewegung abgestimmt, wobei die anderen Bewegungsobjekte mit geringeren Geschwindigkeiten bewegt werden.
- Alle koordinierten Bewegungsinstruktionen müssen mit derselben ID-Nummer gekennzeichnet werden (siehe Instruktion `MoveL`).

Programmtasks können zum Testen über das FlexPendant-Taskauswahlfeld ausgeschlossen werden. Die Instruktion `SyncMoveOn` kann auch mit der geringeren Anzahl an Programmtasks, auch mit nur einer Programmtask, abgearbeitet werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_SYNCMOVEON`	Timeout von `SyncMoveOn`.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `SyncMoveOn` sind unten aufgeführt.

Beispiel 1:

```
!Program example in task T_ROB1
PERS tasks task_list{2} := [["T_ROB1"], ["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WOBJ:= rob2_obj;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.283 SyncMoveOn - Beginn von koordinierten Synchronbewegungen

RW-MRS Synchronized

Fortsetzung

```
  MoveL * \ID:=20, v100, fine, tcp1 \WOBJ:= rob2_obj;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
!Program example in task T_ROB2
PERS tasks task_list{2} := [["T_ROB1"], ["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, obj2;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, obj2;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, obj2;
  MoveL * \ID:=20, v100, fine, obj2;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
```

Zunächst warten die Programmtasks `T_ROB1` und `T_ROB2` bei `WaitSyncTask` mit der Identität `sync1` aufeinander. Sie sind für die vorangegangenen Bewegungen mit einer Zonenbahn programmiert, um die Zyklusdauer zu verkürzen.

Anschließend warten die Programmtasks bei `SyncMoveOn` mit der Identität `sync2` aufeinander. Sie sind mit einem erforderlichen Stopppunkt für die vorangegangenen Bewegungen programmiert. Danach wird die Bewegungsplanung für die betreffenden Programmtasks in den synchronisierten Modus versetzt.

Anschließend bewegt `T_ROB2` das `obj2` zum `ID` -Punkt`10` und `20` im Welt-Koordinatensystem, während `T_ROB1` den `tcp1` zum `ID` -Punkt `10` und `20` auf dem sich bewegenden Objekt `obj2` bewegt.

Beispiel 2:

```
!Program example with use of time-out function
VAR syncident sync3;

...
SyncMoveOn sync3, task_list \TimeOut :=60;
...
ERROR
  IF ERRNO = ERR_SYNCMOVEON THEN
    RETRY;
  ENDIF
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Programmtask wartet darauf, dass die Instruktion `SyncMoveOn` **für die Programmtask** `T_ROB2` **denselben Synchronisierungspunkt** `sync3` **erreicht. Nach einer Wartezeit von 60 Sekunden wird die Fehlerbehandlungsroutine mit** `ERRNO` **gleich** `ERR_SYNCMOVEON` **aufgerufen. Dann wird die Instruktion** `SyncMoveOn` **nochmals aufgerufen und die Wartezeit um 60 Sekunden verlängert.**

Beispiel 3 - Programmbeispiel mit drei Tasks

```
!Program example in task T_ROB1
PERS tasks task_list1 {2} :=[["T_ROB1"], ["T_ROB2"]];
PERS tasks task_list2 {3} :=[["T_ROB1"], ["T_ROB2"], ["T_ROB3"]];
VAR syncident sync1;
...
VAR syncident sync5;

...
  SyncMoveOn sync1, task_list1;
  ...
  SyncMoveOff sync2;
  WaitSyncTask sync3, task_list2;
  SyncMoveOn sync4, task_list2;
  ...
  SyncMoveOff sync5;
...

!Program example in task T_ROB2

PERS tasks task_list1 {2} := [["T_ROB1"], ["T_ROB2"]];
PERS tasks task_list2 {3} := [["T_ROB1"], ["T_ROB2"], ["T_ROB3"]];
VAR syncident sync1;
...
VAR syncident sync5;

...
  SyncMoveOn sync1, task_list1;
  ...
  SyncMoveOff sync2;
  WaitSyncTask sync3, task_list2;
  SyncMoveOn sync4, task_list2;
  ...
  SyncMoveOff sync5;
...

!Program example in task T_ROB3

PERS tasks task_list2 {3} := [["T_ROB1"], ["T_ROB2"], ["T_ROB3"]];
VAR syncident sync3;
VAR syncident sync4;
VAR syncident sync5;

...
  WaitSyncTask sync3, task_list2;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.283 SyncMoveOn - Beginn von koordinierten Synchronbewegungen

RW-MRS Synchronized

Fortsetzung

```
    SyncMoveOn sync4, task_list2;
    ...
    SyncMoveOff sync5;
...
```

In diesem Beispiel führen zunächst die Programmtasks T_ROB1 und T_ROB2 synchronisierte Bewegungen aus und T_ROB3 bewegt sich unabhängig. Im weiteren Verlauf des Programms führen alle drei Tasks synchronisierte Bewegungen aus. Um zu verhindern, dass die Instruktion SyncMoveOn in T_ROB3 abgearbeitet wird, bevor die erste Synchronisierung von T_ROB1 und T_ROB2 beendet wurde, wird die Instruktion WaitSyncTask verwendet.

Einschränkungen

Die Instruktion SyncMoveOn kann nur abgearbeitet werden, wenn sich alle betreffenden Roboter am Stopppunkt im Stillstand befinden.

Es kann jeweils nur eine koordinierte synchronisierte Bewegungsgruppe aktiv sein.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata fine) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

SyncMoveOn kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
SyncMoveOn
  [ SyncID ':=' ] < variable (VAR) of syncident> ','
  [ TaskList ':=' ] < persistent array {*} (PERS) of tasks> ','
  [ '\' TimeOut ':=' < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Identität für Synchronisierungspunkte	*syncident - Identität für Synchronisierungspunkte auf Seite 1814*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*
Festlegen von unabhängigen Bewegungen	*SyncMoveUndo - Festlegen von unabhängigen Bewegungen auf Seite 873*
Testen, ob im synchronisierten Modus	*IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus auf Seite 1385*
MultiMove-System mit Option Coordinated Robots	*Anwendungshandbuch - MultiMove*
Warten auf synchronisierte Tasks	*WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten auf Seite 1089*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.284 SyncMoveResume - Festlegen von synchronisierten koordinierten Bewegungen

Verwendung

`SyncMoveResume` wird verwendet, um vom unabhängigen Bewegungsmodus zu synchronisierten Bewegungen zurückzuwechseln. Die Instruktion kann nur auf der `StorePath`-Ebene verwendet werden, beispielsweise nachdem `StorePath \KeepSync` ausgeführt wurde und sich das System nach der Abarbeitung von `SyncMoveSuspend` im unabhängigen Bewegungsmodus befindet. Um die Instruktion verwenden zu können, muss sich das System vor der Abarbeitung der Instruktionen `StorePath` und `SyncMoveSuspend` im synchronisierten Bewegungsmodus befunden haben.

Die Instruktion `SyncMoveResume` kann nur in einem *MultiMove*-System mit die Optionen *Coordinated Robots* und *Path Recovery* und nur in als *Motion Task* definierten Programmtasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SyncMoveResume`:

Beispiel 1:

```
ERROR
  StorePath \KeepSync;
  ! Save position
  p11 := CRobT(\Tool:=tool2);
  ! Move in synchronized motion mode
  MoveL p12\ID:=111, v50, fine, tool2;
  SyncMoveSuspend;
  ! Move in independent mode somewhere, e.g. to a cleaning station
  p13 := CRobT();
  MoveL p14, v100, fine, tool2;
  ! Do something at cleaning station
  MoveL p13, v100, fine, tool2;
  SyncMoveResume;
  ! Move in synchronized motion mode back to start position p11
  MoveL p11\ID:=112, v50, fine, tool2;
  RestoPath;
  StartMove;
  RETRY;
```

Ein behebbarer Fehler tritt auf. Das System verbleibt im synchronisierten Modus und es wird eine synchronisierte Bewegung zu einem Punkt ausgeführt, beispielsweise eine Rückwärtsbewegung auf der Bahn. Anschließend wird eine unabhängige Bewegung zu einer Reinigungsstation ausgeführt. Anschließend wird der Roboter an den Punkt zurückbewegt, an dem der Fehler aufgetreten ist, und das Programm wird dort fortgesetzt, wo es durch den Fehler unterbrochen wurde.

Programmabarbeitung

`SyncMoveResume` erzwingt die Wiederaufnahme des synchronisierten Modus, wenn sich das System auf `StorePath`-Ebene im unabhängigen Modus befindet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.284 SyncMoveResume - Festlegen von synchronisierten koordinierten Bewegungen

RW-MRS Synchronized
Fortsetzung

`SyncMoveResume` ist in allen Tasks erforderlich, die im synchronisierten Bewegungsmodus abgearbeitet wurden, bevor das System in den unabhängigen Bewegungsmodus versetzt wurde. Falls eine Bewegungstask eine `SyncMoveResume`-Instruktion abarbeitet, wartet diese Task, bis alle Tasks, die sich zuvor im synchronisierten Bewegungsmodus befunden haben, eine `SyncMoveResume`-Instruktion abgearbeitet haben. Danach fahren die betreffenden Programmtasks mit der Abarbeitung fort.

Einschränkungen

Die Instruktion `SyncMoveResume` kann nur zum Zurückwechseln in den synchronisierten Bewegungsmodus und nur auf `StorePath`-Ebene verwendet werden.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`SyncMoveResume` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart,Reset oder Step.

Syntax

```
SyncMoveResume ';'
```

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*
Testen, ob im synchronisierten Modus	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Speichern der Bahn	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Wiederherstellen der Bahn	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Unterbrechen synchronisierter Bewegungen	*SyncMoveSuspend - Festlegen von unabhängigen teilkoordinierten Bewegungen auf Seite 871*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.285 SyncMoveSuspend - Festlegen von unabhängigen teilkoordinierten Bewegungen

Verwendung

`SyncMoveSuspend` wird verwendet, um den synchronisierten Bewegungsmodus auszusetzen und das System in den unabhängigen teilkoordinierten Bewegungsmodus zu versetzen. Die Instruktion kann nur auf `StorePath`-Ebene verwendet werden, also z. B. nachdem `StorePath` oder `StorePath \KeepSync` abgearbeitet wurde und sich das System im synchronisierten Bewegungsmodus befindet.

Die Instruktion `SyncMoveSuspend` kann nur in einem *MultiMove System* mit die Optionen *Coordinated Robots* und *Path Recovery* und nur in als *Motion Task* definierten Programmtasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SyncMoveSuspend`:

Beispiel 1:

```
ERROR
  StorePath \KeepSync;
  ! Save position
  p11 := CRobT(\Tool:=tool2);
  ! Move in synchronized motion mode
  MoveL p12\ID:=111, v50, fine, tool2;
  SyncMoveSuspend;
  ! Move in independent mode somewhere, e.g. to a cleaning station
  p13 := CRobT();
  MoveL p14, v100, fine, tool2;
  ! Do something at cleaning station
  MoveL p13, v100, fine, tool2;
  SyncMoveResume;
  ! Move in synchronized motion mode back to start position p11
  MoveL p11\ID:=112, v50, fine, tool2;
  RestoPath;
  StartMove;
  RETRY;
```

Ein behebbarer Fehler tritt auf. Das System verbleibt im synchronisierten Modus und es wird eine synchronisierte Bewegung zu einem Punkt ausgeführt, beispielsweise eine Rückwärtsbewegung auf der Bahn. Anschließend wird eine unabhängige Bewegung zu einer Reinigungsstation ausgeführt. Anschließend wird der Roboter an den Punkt zurückbewegt, an dem der Fehler aufgetreten ist, und das Programm wird dort fortgesetzt, wo es durch den Fehler unterbrochen wurde.

Programmabarbeitung

`SyncMoveSuspend` erzwingt das Zurücksetzen von synchronisierten koordinierten Bewegungen und das Versetzen des Systems in den unabhängigen teilkoordinierten Bewegungsmodus.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.285 SyncMoveSuspend - Festlegen von unabhängigen teilkoordinierten Bewegungen

RW-MRS Synchronized

Fortsetzung

`SyncMoveSuspend` ist für alle synchronisierten Bewegungstasks erforderlich, um das System in den unabhängigen teilkoordinierten Bewegungsmodus zu versetzen. Falls eine Bewegungstask eine `SyncMoveSuspend`-Instruktion abarbeitet, wartet diese Task, bis die anderen Tasks eine `SyncMoveSuspend`-Instruktion abgearbeitet haben.

Nach der Abarbeitung von `SyncMoveSuspend` in allen beteiligten Tasks befindet sich das System weiterhin im teilkoordinierten Modus, falls weiterhin ein koordiniertes Werkobjekt verwendet wird. Anderenfalls befindet es sich um unabhängigen Modus. Im teilkoordinierten Modus wird empfohlen, immer mit einer Bewegung der mechanischen Einheit zu beginnen, die das Anwender-Koordinatensystem steuert, bevor `WaitSyncTask` in allen beteiligten Tasks abgearbeitet wird.

Einschränkungen

Die Instruktion `SyncMoveSuspend` unterbricht den synchronisierten Modus nur auf `StorePath`-Ebene. Nach der Rückkehr von der `StorePath`-Ebene wird das System wieder in den Modus versetzt, in dem es sich vor `StorePath` befunden hat.

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`SyncMoveSuspend` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart,Reset oder Step.

Syntax

```
SyncMoveSuspend ';'
```

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*
Testen, ob im synchronisierten Modus	*IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus auf Seite 1385*
Speichern der Bahn	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Wiederherstellen der Bahn	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Wiederaufnehmen synchronisierter Bewegungen	*SyncMoveResume - Festlegen von synchronisierten koordinierten Bewegungen auf Seite 869*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.286 SyncMoveUndo - Festlegen von unabhängigen Bewegungen

Verwendung

`SyncMoveUndo` wird verwendet, um synchronisierte koordinierte Bewegungen zurückzusetzen und das System in den unabhängigen Bewegungsmodus zu versetzen.

Die Instruktion `SyncMoveUndo` kann nur in einem *MultiMove*-System mit der Option *Coordinated Robots* und nur in als `Motion Task` definierten Programmtasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `SyncMoveUndo`:

Beispiel 1:

Programmbeispiel in Task `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
PROC main()

...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WOBJ:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WOBJ:= rob2_obj;
  SyncMoveOff sync3;
UNDO
  SyncMoveUndo;
ENDPROC
```

Wenn die Programmabarbeitung in der Prozedur `syncmove` angehalten und der Programmzeiger aus der Prozedur `syncmove` bewegt wird, werden alle Instruktionen in der `UNDO`-Routine ausgeführt. In diesem Beispiel wird die Instruktion `SyncMoveUndo` abgearbeitet und das System wird in den unabhängigen Bewegungsmodus versetzt.

Programmabarbeitung

Erzwungenes Zurücksetzen von synchronisierten koordinierten Bewegungen und Versetzen des Systems in den unabhängigen Bewegungsmodus

Es reicht aus, `SyncMoveUndo` in einer Programmtask auszuführen, um das gesamte System in den unabhängigen Bewegungsmodus zu versetzen. Die Instruktion kann

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.286 SyncMoveUndo - Festlegen von unabhängigen Bewegungen
RobotWare - OS
Fortsetzung

mehrmals ohne Fehler ausgeführt werden, wenn sich das System bereits im unabhängigen Modus befindet.

Das System wird auch

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Syntax

```
SyncMoveUndo ';'
```

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Identität für Synchronisierungspunkte	*syncident - Identität für Synchronisierungspunkte auf Seite 1814*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*
Testen, ob im synchronisierten Modus	*IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus auf Seite 1385*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.287 SyncToSensor - Synchr. mit Sensor

Verwendung

SyncToSensor wird dazu verwendet, die Synchronisierung der Roboterbewegung mit der Sensorbewegung zu starten oder zu stoppen.

Grundlegende Beispiele

Grundlegende Beispiele für die Instruktion `SyncToSensor` sind unten aufgeführt.

Beispiel 1:

```
WaitSensor Ssync1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
SyncToSensor Ssync1\On;
MoveL *, v1000, z20, tool, \WObj:=wobj0;
MoveL *, v1000, z20, tool, \WObj:=wobj0;
SyncToSensor Ssync1\Off;
```

Argumente

```
SyncToSensor MechUnit [\MaxSync] [\On] | [\Off]
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Die sich bewegende mechanische Einheit, auf die sich die Roboterposition in der Instruktion bezieht.

`[\MaxSync]`

Datentyp: `num`

Der Roboter bewegt sich synchronisiert zum Sensor, bis der Sensor die Position `MaxSync` erreicht. Danach bewegt sich der Roboter unsynchronisiert mit Programmgeschwindigkeit. Wurde der optionale Parameter `MaxSync` nicht definiert, bewegt sich der Roboter synchronisiert, bis die Instruktion `SyncToSensor Ssync1\Off` ausgeführt wird.

`[\On]`

Datentyp: `switch`

Der Roboter bewegt sich nach der Instruktion mit dem Argument `\On` synchronisiert zum Sensor.

`[\Off]`

Datentyp: `switch`

Der Roboter bewegt sich nach der Instruktion mit dem Argument `\Off` unsynchronisiert zum Sensor.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.287 SyncToSensor - Synchr. mit Sensor

Machine Synchronization
Fortsetzung

Programmabarbeitung

`SyncToSensor Ssync1 \On` bedeutet, dass sich der Roboter synchronisiert zum Sensor `Ssync1` bewegt. Daher passiert der Roboter das programmierte `robtarget` zur gleichen Zeit, zu der der Sensor die externe Position passiert, die in `robtarget` gespeichert ist.

`SyncToSensor Ssync1 \Off` bedeutet, dass die zum Sensor synchronisierte Bewegung des Roboters endet.

Einschränkungen

Wird die Instruktion `SyncToSensor Ssync1 \On` ausgegeben, wenn der Sensor nicht über `WaitSensor` verbunden wurde, hält der Roboter an.

Syntax

```
SyncToSensor
  [ MechUnit ':=' ] < variable (VAR) of mecunit >
  [ \MaxSync] [ '\' On] | [ '\' Off] ';'
```

Weitere Informationen

Information über	Siehe
Start der synchronisierten Sensorüberwachung	*SupSyncSensorOn - Start der synchronisierten Sensorüberwachung auf Seite 855*
Synchr. mit Sensor	*SyncToSensor - Synchr. mit Sensor auf Seite 875*
Wartet auf Verbindung zu Sensor	*WaitSensor - Auf die Sensorverbindung warten auf Seite 1085*
Ablegen des Objekts am Sensor	*DropSensor - Ablegen des Objekts am Sensor auf Seite 191*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.288 SystemStopAction - Anhalten des Robotersystems

Verwendung

Mit `SystemStopAction` kann das Robotersystem auf verschiedene Weise angehalten werden, abhängig vom Schweregrad des Fehlers oder Problems.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `SystemStopAction`:

Beispiel 1:

```
SystemStopAction \Stop;
```

Hierdurch werden die Programmabarbeitung und die Roboterbewegungen in allen Bewegungstasks angehalten. Vor dem Neustart der Programmabarbeitung ist keine besondere Maßnahme erforderlich.

Beispiel 2:

```
SystemStopAction \StopBlock;
```

Hierdurch werden die Programmabarbeitung und die Roboterbewegungen in allen Bewegungstasks angehalten. Alle Programmzeiger müssen verschoben werden, bevor die Programmabarbeitung neu gestartet werden kann.

Beispiel 3

```
SystemStopAction \Halt;
```

Dies führt in allen Bewegungstasks zum Zustand MOTORS OFF sowie zum Anhalten von Programmabarbeitung und Roboterbewegungen. MOTORS ON muss ausgeführt werden, bevor die Programmabarbeitung neu gestartet werden kann.

Argumente

```
SystemStopAction [\Stop] [\StopBlock] [\Halt]
```

`[\Stop]`

Datentyp: `switch`

Mit `\Stop` werden die Programmabarbeitung und die Roboterbewegungen in allen Bewegungstasks angehalten. Vor dem Neustart der Programmabarbeitung ist keine besondere Maßnahme erforderlich.

`[\StopBlock]`

Datentyp: `switch`

Mit `\StopBlock` werden die Programmabarbeitung und die Roboterbewegungen in allen Bewegungstasks angehalten. Alle Programmzeiger müssen verschoben werden, bevor die Programmabarbeitung neu gestartet werden kann.

`[\Halt]`

Datentyp: `switch`

Mit `\Halt` werden in allen Bewegungstasks die Motoren ausgeschaltet und die Programmabarbeitung sowie die Roboterbewegungen angehalten. MOTORS ON muss ausgeführt werden, bevor die Programmabarbeitung neu gestartet werden kann.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.288 SystemStopAction - Anhalten des Robotersystems

RobotWare - OS

Fortsetzung

Programmabarbeitung

Mit `SystemStopAction` kann das Robotersystem auf verschiedene Weise angehalten werden, abhängig vom Schweregrad des Fehlers oder Problems. Die Programmabarbeitung wird in der ausführenden Task angehalten, wenn es sich bei der Task um eine normale Task handelt.

Wenn `SystemStopAction` in einer statischen oder semistatischen Task ausgeführt wird, wird die Programmausführung für alle normalen Tasks außer dieser Task angehalten. Weitere Informationen über die Deklaration von Tasks finden Sie in der Dokumentation zu Systemparametern.

Einschränkungen

Wenn der Roboter während `SystemStopAction \StopBlock` eine kreisförmige Bewegung ausführt, müssen Programmzeiger und Roboter an den Anfang der kreisförmigen Bewegung versetzt werden, bevor die Programmabarbeitung neu gestartet wird.

Syntax

```
SystemStopAction
  [ '\'Stop ]
  | [ '\'StopBlock ]
  | [ '\'Halt ]';'
```

Weitere Informationen

Information über	Siehe
Anhalten der Programmabarbeitung	*Stop - Stoppt die Programmabarbeitung auf Seite 833*
Beenden der Programmabarbeitung	*EXIT - Beenden der Programmabarbeitung auf Seite 213*
Nur Roboterbewegungen anhalten	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Schreiben einer Fehlermeldung	*ErrLog - Schreiben einer Fehlermeldung auf Seite 202*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.289 TEST - Abhängig vom Wert eines Ausdrucks ...

Verwendung

TEST wird verwendet, wenn verschiedene Instruktionen abhängig vom Wert eines Ausdrucks oder von Daten abzuarbeiten sind.

Bei nur wenigen Alternativen eignet sich auch die Instruktion IF..ELSE.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion TEST:

Beispiel 1:

```
TEST reg1
CASE 1,2,3 :
  routine1;
CASE 4 :
  routine2;
DEFAULT :
  TPWrite "Illegal choice";
  Stop;
ENDTEST
```

Ausführen verschiedener Instruktionen abhängig vom Wert des Ausdrucks reg1. Ist der Wert 1, 2 oder 3, dann wird routine1 nicht ausgeführt. Ist der Wert 4, dann wird routine2 ausgeführt. Andernfalls wird eine Fehlermeldung gedruckt und die Abarbeitung stoppt.

Argumente

```
TEST Test data {CASE Test value {, Test value} : ...} [ DEFAULT:
     ...] ENDTEST
```

Test data

Datentyp: All

Die Daten oder der Ausdruck, mit dem der Wert verglichen wird.

Test value

Datentyp: Wie test data

Der Wert, den die Testdaten aufweisen müssen, damit die zugehörigen Instruktionen abgearbeitet werden.

Programmabarbeitung

Die Testdaten werden mit den Testwerten in der ersten CASE-Bedingung verglichen. Wenn der Vergleich wahr ist, werden die zugehörigen Instruktionen abgearbeitet. Anschließend fährt die Programmabarbeitung mit der Instruktion fort, die auf ENDTEST folgt.

Wenn die erste CASE-Bedingung nicht erfüllt wird, werden andere CASE-Bedingungen getestet usw. Wenn keine der Bedingungen erfüllt ist, werden die Instruktionen abgearbeitet, die mit DEFAULT verbunden sind (falls vorhanden).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.289 TEST - Abhängig vom Wert eines Ausdrucks ...

RobotWare - OS

Fortsetzung

Syntax

```
TEST <expression>
{ CASE <test value> { ',' <test value> } ':'
  <statement list> }
[ DEFAULT ':'
  <statement list> ]
ENDTEST
```

Weitere Informationen

Information über	Siehe
Ausdrücke	*Technisches Referenzhandbuch - RAPID Instruktionen, Funktionen und Datentypen*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.290 TestSignDefine - Definieren eines Prüfsignals

Verwendung

`TestSignDefine` wird verwendet, um ein Prüfsignal für das Roboterbewegungssystem zu definieren.

Ein Prüfsignal spiegelt kontinuierlich einen angegebenen Bewegungsdatenstrom, z. B. die Drehmoment-Referenz einer angegebenen Achse. Der tatsächliche Wert zu einem bestimmten Zeitpunkt kann in RAPID mit der Funktion `TestSignRead` abgelesen werden.

Nur Prüfsignale für externe Achsen können erreicht werden. Prüfsignale sind auch auf Anforderung für die Roboterachsen verfügbar, und nicht vordefinierte Prüfsignale sind für externe Achsen verfügbar.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TestSignDefine`:

Beispiel 1:

```
TestSignDefine 1, resolver_angle, Orbit, 2, 0.1;
```

Prüfsignal `resolver_angle`, verbunden mit Kanal `1` liefert den Wert des Resolverwinkels für die externe Achse `2` am `orbit`-Manipulator , mit einer Abtastrate von 100 ms.

Argumente

```
TestSignDefine Channel SignalId MechUnit Axis SampleTime
```

`Channel`

Datentyp: `num`

Die Kanalnummern 1-12 sind für das Prüfsignal vorgesehen. Dieselbe Nummer muss in der Funktion `TestSignRead` zum Lesen des aktuellen Prüfsignalwerts verwendet werden.

`SignalId`

Datentyp: `testsignal`

Der Name oder die Nummer des Prüfsignals. Siehe vordefinierte Konstanten, die im Datentyp `testsignal` beschrieben werden.

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Achsennummer in der mechanischen Einheit.

`SampleTime`

Datentyp: `num`

Abtastzeit in Sekunden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.290 TestSignDefine - Definieren eines Prüfsignals

RobotWare - OS

Fortsetzung

Für Abtastzeiten < 0,004 s gibt die Funktion `TestSignRead` den Mittelwert der letzten verfügbaren internen Abtastungen zurück wie in nachstehender Tabelle gezeigt.

Abtastzeit in Sekunden	Ergebnis von TestSignRead
0	Mittelwert der letzten 8 Abtastungen, alle 0,5 ms generiert
0.001	Mittelwert der letzten 4 Abtastungen, alle 1 ms generiert
0.002	Mittelwert der letzten 2 Abtastungen, alle 2 ms generiert
Größer oder gleich 0,004	Momentaner Wert, generiert in der angegebenen Abtastzeit
0.1	Momentaner Wert, generiert in der angegebenen Abtastzeit von 100 ms

Programmabarbeitung

Die Definition des Prüfsignals wird aktiviert und das Robotersystem startet die Abtastung des Prüfsignals.

Die Abtastung des Prüfsignals ist aktiv bis:

- eine neue `TestSignDefine`-Instruktion für den aktuellen Kanal abgearbeitet wird,
- alle Prüfsignale mit der Abarbeitung der Instruktion `TestSignReset` deaktiviert werden,
- Bei einem **Neustart** des Systems werden alle Prüfsignale deaktiviert.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AXIS_PAR`	Es ist ein Fehler im Parameter `Axis` aufgetreten.
`ERR_UNIT_PAR`	Es ist ein Fehler im Parameter `MechUnit` aufgetreten.

Syntax

```
TestSignDefine
  [ Channel ':=' ] < expression (IN) of num>' ,'
  [ SignalId' :=' ] < expression (IN) of testsignal> ','
  [ MechUnit' :=' ] < variable (VAR) of mecunit> ','
  [ Axis ':=' ] < expression (IN) of num> ','
  [ SampleTime' :=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Prüfsignal	*testsignal – Prüfsignal auf Seite 1821*
Lesen eines Prüfsignals	*TestSignRead - Lesen eines Prüfsignalwerts auf Seite 1554*
Zurücksetzen eines Prüfsignals	*TestSignReset - Zurücksetzen aller Prüfsignaldefinitionen auf Seite 883*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.291 TestSignReset - Zurücksetzen aller Prüfsignaldefinitionen

Verwendung

`TestSignReset` wird verwendet, um alle zuvor definierten Prüfsignale zurückzusetzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TestSignReset`:

Beispiel 1:

```
TestSignReset;
```

Deaktiviert alle zuvor definierten Prüfsignale.

Programmabarbeitung

Die Definitionen aller Prüfsignale werden deaktiviert und das Robotersystem stoppt die Abtastung jeglicher Prüfsignale.

Die Abtastung der definierten Prüfsignale ist aktiv bis:

- Ein **Neustart** des Systems
- Abarbeitung der Instruktion `TestSignReset`

Syntax

```
TestSignReset';'
```

Weitere Informationen

Information über	Siehe
Prüfsignal	*testsignal – Prüfsignal auf Seite 1821*
Definieren eines Prüfsignals	*TestSignDefine - Definieren eines Prüfsignals auf Seite 881*
Lesen eines Prüfsignals	*TestSignRead - Lesen eines Prüfsignalwerts auf Seite 1554*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.292 TextTabInstall - Installieren einer Texttabelle

Verwendung

Mit `TextTabInstall` wird eine Texttabelle im System installiert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TextTabInstall`.

Beispiel 1:

```
! System Module with Event Routine to be executed at event
! POWER ON, RESET or START

PROC install_text()
  IF TextTabFreeToUse("text_table_name") THEN
    TextTabInstall "HOME:/text_file.xml";
  ENDIF
ENDPROC
```

Wenn die Ereignisroutine `install_text` ausgeführt wird, gibt die Funktion `TextTabFreeToUse` den TRUE zurück und die Textdatei `text_file.xml` wird im System installiert. Danach können die installierten Zeichenfolgen mit den Funktionen `TextTabGet` und `TextGet` vom System in RAPID abgerufen werden.

Bei der nächsten Abarbeitung der Ereignisroutine `install_text` gibt die Funktion `TextTabFreeToUse` FALSE zurück und die Installation wird **nicht** wiederholt.

Argumente

```
TextTabInstall File
```

`File`

Datentyp: `string`

Der Dateipfad und der Dateiname der Datei, die Textstrings enthält, die im System installiert werden sollen.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEOPEN`	Die in der Instruktion `TextTabInstall` angegebene Datei kann nicht geöffnet werden.

Einschränkungen

Einschränkungen für die Installation von Texttabellen (Textquellen) im System:

- Es ist nicht möglich, dieselbe Texttabelle mehrmals im System zu installieren.
- Es ist nicht möglich, eine einzelne Texttabelle vom System zu deinstallieren (herauszunehmen). Der einzige Weg, Texttabellen vom System zu deinstallieren, ist ein Neustart der Steuerung mithilfe des Neustartmodus

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

System zurücksetzen. Alle Texttabellen (System- und benutzerdefinierte Tabellen) werden dann deinstalliert.

Syntax

```
TextTabInstall
  [File ':='] <expression (IN) of string>';'
```

Weitere Informationen

Information über	Siehe
Test, ob Texttabelle frei ist	*TextTabFreeToUse - Prüfen, ob Texttabelle frei ist auf Seite 1559*
Format von Textdateien	*Technical reference manual - RAPID kernel*
Abrufen der Texttabellennummer	*TextTabGet - Abrufen der Texttabellennummer auf Seite 1561*
Abrufen von Text aus Systemtext-tabellen	*TextGet - Abrufen von Text aus Systemtexttabellen auf Seite 1556*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.293 TPErase - Löschen des Texts am FlexPendant

Verwendung

`TPErase` (*FlexPendant Erase*) wird verwendet, um die Anzeige des FlexPendant zu löschen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TPErase`:

Beispiel 1:

```
TPErase;
TPWrite "Execution started";
```

Die Anzeige des FlexPendant wird gelöscht, bevor `Execution started` geschrieben wird.

Programmabarbeitung

Von der Anzeige des FlexPendant wird sämtlicher Text entfernt. Beim nächsten Schreiben von Text beginnt dieser mit der ersten Zeile der Anzeige.

Syntax

```
TPErase;
```

Weitere Informationen

Information über	Siehe
Schreiben auf das FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.294 TPReadDnum - Lesen eines numerischen Werts am FlexPendant

Verwendung

`TPReadDnum` (*FlexPendant Read Numerical*) wird verwendet, um eine Zahl vom FlexPendant zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TPReadDnum`:

Beispiel 1:

```
VAR dnum value;

TPReadDnum value, "How many units should be produced?";
```

Der Text `How many units should be produced?` wird auf dem FlexPendant-Display ausgegeben. Die Programmabarbeitung wartet, bis eine Zahl über den numerischen Tastenblock am FlexPendant eingegeben wurde. Diese Zahl wird in `value` gespeichert.

Argumente

```
TPReadDnum TPAnswer TPText [\MaxTime][\DIBreak] [\DIPassive]
     [\DOBreak] [\DOPassive] [\PersBoolBreak] [\PersBoolPassive]
     [\BreakFlag]
```

`TPAnswer`

Datentyp: `dnum`

Die Variable, für die die Zahleneingabe über das FlexPendant zurückgegeben wird.

`TPText`

Datentyp: `string`

Der Informationstext, der auf das FlexPendant geschrieben werden soll (maximal 80 Zeichen mit 40 Zeichen/Zeile).

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer, für die die Programmabarbeitung wartet. Wenn innerhalb dieser Zeitdauer keine Zahl eingegeben wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann geprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Signal, das den Bedienerdialog unterbrechen kann. Falls keine Zahl eingegeben wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.294 TPReadDnum - Lesen eines numerischen Werts am FlexPendant
RobotWare - OS
Fortsetzung

[\DIPassive]

Digital Input Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DIBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DIBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Das digitale Signal unterstützt die Beendigungsanforderung durch andere Tasks. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DOBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolBreak]

Persistent Boolean Break

Datentyp: bool

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolPassive]

Persistent Boolean Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument PersBoolBreak verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

verwendet wird), wenn die persistente boolesche Variable PersBoolBreak auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

Programmabarbeitung

Der Informationstext wird immer in eine neue Zeile geschrieben. Wenn die Anzeige mit Text gefüllt ist, wird der Textbereich um jeweils eine Zeile nach oben geschoben. Es können bis zu 7 Zeilen über dem neu geschriebenen Text vorhanden sein.

Die Programmabarbeitung wartet, bis eine Zahl am numerischen Tastenblock eingegeben wird (gefolgt von der Eingabetaste oder OK) oder die Instruktion durch einen Timeout oder eine Signalaktion unterbrochen wird.

Siehe auch TPReadFK für die Beschreibung der gleichzeitigen Anforderung TPReadFK oder TPReadDnum am FlexPendant aus derselben oder einer anderen Programmtask.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_MAXTIME	Timeout (Parameter \MaxTime) vor einer Bedienereingabe.
ERR_TP_DIBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Eingang unterbrochen. Vor einer Bedienereingabe wurde ein digitaler Eingang gesetzt (Parameter \DIBreak).
ERR_TP_DOBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Ausgang unterbrochen. Vor einer Bedienereingabe wurde ein digitaler Ausgang gesetzt (Parameter \DOBreak).
ERR_TP_NO_CLIENT	Es ist kein Client für Interaktionen vorhanden bei Verwendung einer Instruktion vom FlexPendant.
ERR_TP_PERSBOOLBREAK	Eine Read-Instruktion vom FlexPendant wurde durch eine persistene boolesche Variable unterbrochen. Vor einer Bedienereingabe wurde eine persistente boolesche Variable (Parameter \PersBoolBreak) geändert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.294 TPReadDnum - Lesen eines numerischen Werts am FlexPendant

RobotWare - OS

Fortsetzung

Syntax

```
TPReadDnum
  [TPAnswer':='] <var or pers (INOUT) of dnum>','
  [TPText':='] <expression (IN) of string>
  ['\'MaxTime':=' <expression (IN) of num>]
  ['\'DIBreak':=' <variable (VAR) of signaldi>]
  ['\'DIPassive]
  ['\'DOBreak':=' <variable (VAR) of signaldo>]
  ['\'DOPassive]
  ['\'PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\'PersBoolPassive]
  ['\'BreakFlag':=' <var or pers (INOUT) of errnum>] ';'
```

Weitere Informationen

Information über	Siehe
Schreiben und Lesen am FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*
Eingeben eines numerischen Werts am FlexPendant	*Bedienungsanleitung - IRC5 mit FlexPendant*
Beispiele für die Verwendung der Argumente MaxTime, DIBreak und BreakFlag	*TPReadFK - Lesen der Funktionstasten auf Seite 891*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.295 TPReadFK - Lesen der Funktionstasten

Verwendung

`TPReadFK` (*FlexPendant Read Function Key*) wird verwendet, um Beschriftungen an den Funktionstasten anzubringen und zu ermitteln, welche Taste gedrückt wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TPReadFK`:

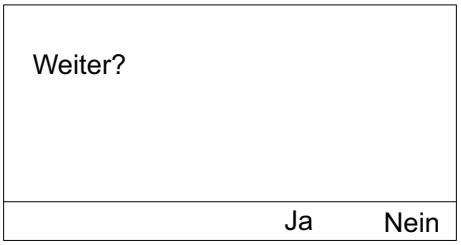
Siehe auch *Weitere Beispiele auf Seite 894*.

Beispiel 1:

```
TPReadFK reg1, "More?", stEmpty, stEmpty, stEmpty, "Yes", "No";
```

Der Text `More?` wird auf der Anzeige des FlexPendant ausgegeben und die Funktionstasten 4 und 5 werden mit der Zeichenfolge `Yes` bzw. `No` aktiviert (siehe Abbildung unten). Die Programmabarbeitung wartet, bis Funktionstaste 4 oder 5 gedrückt wird. Das heißt, `reg1` wird 4 oder 5 zugewiesen, abhängig davon, welche der Tasten gedrückt wird.

Die Abbildung zeigt, dass der Bediener über die Funktionstasten Informationen eingeben kann.



xx0500002345

Argumente

```
TPReadFK TPAnswer TPText TPFK1 TPFK2 TPFK3 TPFK4 TPFK5 [\MaxTime]
     [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive]
     [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
```

`TPAnswer`

Datentyp: `num`

Die Variable, für die abhängig von der gedrückten Taste der numerische Wert 1...5 zurückgegeben wird. Wenn Funktionstaste 1 gedrückt wird, wird 1 zurückgegeben usw.

`TPText`

Datentyp: `string`

Der Informationstext, der auf die Anzeige geschrieben werden soll (maximal 80 Zeichen mit 40 Zeichen/Zeile).

`TPFKx`

Function key text

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.295 TPReadFK - Lesen der Funktionstasten
RobotWare - OS
Fortsetzung

Der Text, der auf die entsprechende Funktionstaste geschrieben werden soll (maximal 45 Zeichen). TPFK1 ist die Taste am linken Rand.

Funktionstasten ohne Beschriftung werden durch die vordefinierte String-Konstante stEmpty mit einer leeren Zeichenfolge ("") definiert.

[\MaxTime]

Datentyp: num

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn innerhalb dieser Zeitdauer keine Funktionstaste betätigt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_MAXTIME kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

[\DIBreak]

Digital Input Break

Datentyp: signaldi

Das digitale Signal, das den Bedienerdialog unterbrechen kann. Falls keine Funktionstaste betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DIPassive]

Digital Input Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DIBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DIBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Das digitale Signal unterstützt die Beendigungsanforderung durch andere Tasks. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

kein `BreakFlag` verwendet wird), wenn das Signal `DOBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolBreak]`

Persistent Boolean Break

Datentyp: `bool`

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolPassive]`

Persistent Boolean Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `PersBoolBreak` verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn die persistente boolesche Variable `PersBoolBreak` auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\BreakFlag]`

Datentyp: `errnum`

Eine Variable, die den Fehlercode enthält, wenn `MaxTime`, `DIBreak`, `DOBreak` oder `PersBoolBreak` verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK`, `ERR_TP_DOBREAK` und `ERR_TP_PERSBOOLBREAK` können verwendet werden, um die Ursache zu bestimmen.

Programmabarbeitung

Der Informationstext wird immer in eine neue Zeile geschrieben. Wenn die Anzeige mit Text gefüllt ist, wird der Textbereich um jeweils eine Zeile nach oben geschoben. Es können bis zu 7 Zeilen über dem neu geschriebenen Text vorhanden sein.

Text wird über die entsprechenden Funktionstasten geschrieben.

Die Programmabarbeitung wartet, bis eine der aktivierten Funktionstasten gedrückt wird.

Beschreibung von gleichzeitiger Anforderung von `TPReadFK` oder `TPReadNum` am FlexPendant (TP-Anforderung) aus derselben oder einer anderen Programmtask:

- Neue TP-Anforderung von anderen Programmtasks wird nicht beachtet (neue Anforderung in Warteschlange gestellt).
- Neue TP-Anforderung von Interrupt in derselben Programmtask wird beachtet (alte Anforderung in Warteschlange gestellt).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.295 TPReadFK - Lesen der Funktionstasten
RobotWare - OS
Fortsetzung

- Programmstopp wird beachtet (alte Anforderung in Warteschlange gestellt).
- Neue TP-Anforderung in Programmstopp wird beachtet (alte Anforderung in Warteschlange gestellt).

Vordefinierte Daten

```
CONST string stEmpty := "";
```

Die vordefinierte Konstante `stEmpty` kann für Funktionstasten ohne Text verwendet werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
ERR_TP_MAXTIME	Timeout (Parameter `\MaxTime`) vor einer Bedienereingabe.
ERR_TP_DIBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Eingang unterbrochen. Vor einer Bedienereingabe wurde ein digitaler Eingang gesetzt (Parameter `\DIBreak`).
ERR_TP_DOBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Ausgang unterbrochen. Vor einer Bedienereingabe wurde ein digitaler Ausgang gesetzt (Parameter `\DOBreak`).
ERR_TP_NO_CLIENT	Es ist kein Client für Interaktionen vorhanden bei Verwendung einer Instruktion vom FlexPendant.
ERR_TP_PERSBOOLBREAK	Eine Read-Instruktion vom FlexPendant wurde durch eine persistene boolesche Variable unterbrochen. Vor einer Bedienereingabe wurde eine persistente boolesche Variable (Parameter `\PersBoolBreak`) geändert.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TPReadFK` sind unten aufgeführt.

Beispiel 1:

```
VAR errnum errvar;
...
TPReadFK reg1, "Go to service position?", stEmpty, stEmpty, stEmpty,
     "Yes","No"
\MaxTime:= 600
  \DIBreak:= di5\BreakFlag:= errvar;
IF reg1 = 4 OR errvar = ERR_TP_DIBREAK THEN
  MoveL service, v500, fine, tool1;
  Stop;
ENDIF
IF errvar = ERR_TP_MAXTIME EXIT;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Roboter wird an die Wartungsposition bewegt, wenn die vierte Funktionstaste (`"Yes"`) gedrückt oder Eingang 5 aktiviert wird. Geht innerhalb von 10 Minuten keine Antwort ein, wird die Abarbeitung beendet.

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `TPReadFK` häufig, z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
TPReadFK
  [TPAnswer ':='] <var or pers (INOUT) of num>','
  [TPText ':='] <expression (IN) of string>','
  [TPFK1 ':='] <expression (IN) of string>','
  [TPFK2 ':='] <expression (IN) of string>','
  [TPFK3 ':='] <expression (IN) of string>','
  [TPFK4 ':='] <expression (IN) of string>','
  [TPFK5 ':='] <expression (IN) of string>
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\' PersBoolPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]';'
```

Weitere Informationen

Information über	Siehe
Schreiben und Lesen am FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*
Antworten über das FlexPendant	*Bedienungsanleitung - IRC5 mit FlexPendant*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.296 TPReadNum - Lesen eines numerischen Werts am FlexPendant

Verwendung

`TPReadNum` (*FlexPendant Read Numerical*) wird verwendet, um eine Zahl vom FlexPendant zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TPReadNum`:

Siehe auch *Weitere Beispiele auf Seite 899*.

Beispiel 1:

```
TPReadNum reg1, "How many units should be produced?";
```

Der Text `How many units should be produced?` wird auf dem FlexPendant-Display ausgegeben. Die Programmabarbeitung wartet, bis eine Zahl über den numerischen Tastenblock am FlexPendant eingegeben wurde. Diese Zahl wird in `reg1` gespeichert.

Argumente

```
TPReadNum TPAnswer TPText [\MaxTime][\DIBreak] [\DIPassive]
    [\DOBreak] [\DOPassive] [\PersBoolBreak] [\PersBoolPassive]
    [\BreakFlag]
```

`TPAnswer`

Datentyp: `num`

Die Variable, für die die Zahleneingabe über das FlexPendant zurückgegeben wird.

`TPText`

Datentyp: `string`

Der Informationstext, der auf das FlexPendant geschrieben werden soll (maximal 80 Zeichen mit 40 Zeichen pro Zeile).

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer, für die die Programmabarbeitung wartet. Wenn innerhalb dieser Zeitdauer keine Zahl eingegeben wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann geprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Signal, das den Bedienerdialog unterbrechen kann. Falls keine Zahl eingegeben wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\DIPassive]

Digital Input Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DIBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DIBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Das digitale Signal unterstützt die Beendigungsanforderung durch andere Tasks. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DOBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolBreak]

Persistent Boolean Break

Datentyp: bool

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolPassive]

Persistent Boolean Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument PersBoolBreak verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

verwendet wird), wenn die persistente boolesche Variable PersBoolBreak auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

Programmabarbeitung

Der Informationstext wird immer in eine neue Zeile geschrieben. Wenn die Anzeige mit Text gefüllt ist, wird der Textbereich um jeweils eine Zeile nach oben geschoben. Es können bis zu 7 Zeilen über dem neu geschriebenen Text vorhanden sein.

Die Programmabarbeitung wartet, bis eine Zahl am numerischen Tastenblock eingegeben wird (gefolgt von der Eingabetaste oder OK) oder die Instruktion durch einen Timeout oder eine Signalaktion unterbrochen wird.

Siehe auch TPReadFK für die Beschreibung der gleichzeitigen Anforderung TPReadFK oder TPReadNum am FlexPendant aus derselben oder einer anderen Programmtask.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_MAXTIME	Timeout (Parameter \MaxTime) vor einer Bedienereingabe.
ERR_TP_DIBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Eingang unterbrochen. Vor einer Bedienereingabe wurde ein digitaler Eingang gesetzt (Parameter \DIBreak).
ERR_TP_DOBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Ausgang unterbrochen. Vor einer Bedienereingabe wurde ein digitaler Ausgang gesetzt (Parameter \DOBreak).
ERR_TP_NO_CLIENT	Es ist kein Client für Interaktionen vorhanden bei Verwendung einer Instruktion vom FlexPendant.
ERR_TP_PERSBOOLBREAK	Eine Read-Instruktion vom FlexPendant wurde durch eine persistene boolesche Variable unterbrochen. Vor einer Bedienereingabe wurde eine persistente boolesche Variable (Parameter \PersBoolBreak) geändert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TPReadNum` sind unten aufgeführt.

Beispiel 1:

```
TPReadNum reg1, "How many units should be produced?";
FOR i FROM 1 TO reg1 DO
  produce_part;
ENDFOR
```

Der Text `How many units should be produced?` wird am FlexPendant ausgegeben. Die Routine `produce_part` wird dann entsprechend der am FlexPendant eingegebenen Zahl wiederholt.

Syntax

```
TPReadNum
  [TPAnswer':='] <var or pers (INOUT) of num>','
  [TPText':='] <expression (IN) of string>
  ['\'MaxTime':=' <expression (IN) of num>]
  ['\'DIBreak':=' <variable (VAR) of signaldi>]
  ['\'DIPassive]
  ['\'DOBreak':=' <variable (VAR) of signaldo>]
  ['\'DOPassive]
  ['\'PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\'PersBoolPassive]
  ['\'BreakFlag':=' <var or pers (INOUT) of errnum>] ';'
```

Weitere Informationen

Information über	Siehe
Schreiben und Lesen am FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*
Eingeben eines numerischen Werts am FlexPendant	*Bedienungsanleitung - IRC5 mit FlexPendant*
Beispiele für die Verwendung der Argumente MaxTime, DIBreak und BreakFlag	*TPReadFK - Lesen der Funktionstasten auf Seite 891*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.297 TPShow - Wechsel des Fensters am FlexPendant

Verwendung

`TPShow` (*FlexPendant Show*) wird verwendet, um das Fenster des FlexPendant im RAPID-Programm auszuwählen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TPShow`:

Beispiel 1:

```
TPShow TP_LATEST;
```

Das zuletzt am FlexPendant verwendete Fenster vor dem aktuellen Fenster ist nach Abarbeiten dieser Instruktion aktiv.

Argumente

```
TPShow Window
```

`Window`

Datentyp: `tpnum`

Das Fenster TP_LATEST zeigt das Fenster am FlexPendant, das vor dem aktuellen Fenster verwendet wurde.

Programmabarbeitung

Das ausgewählte Fenster des FlexPendant wird aktiviert.

Vordefinierte Daten

```
CONST tpnum TP_LATEST := 2;
```

Syntax

```
TPShow
  [Window':='] <expression (IN) of tpnum> ';'
```

Weitere Informationen

Information über	Siehe
Kommunikation mithilfe des FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*
Nummer des FlexPendant-Fensters	*tpnum - Fensternummer des FlexPendant auf Seite 1830*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.298 TPWrite - Schreiben am FlexPendant

Verwendung

`TPWrite` (*FlexPendant Write*) wird verwendet, um auf das FlexPendant zu schreiben. Auch der Wert von bestimmten Daten kann als Text geschrieben werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `TPWrite`:

Beispiel 1:

```
TPWrite "Execution started";
```

Der Text `Execution started` wird am FlexPendant ausgegeben.

Beispiel 2:

```
TPWrite "No of produced parts="\Num:=reg1;
```

Wenn z. B. `reg1` den Wert 5 enthält, wird der Text `No of produced parts=5` auf dem FlexPendant ausgegeben.

Beispiel 3

```
VAR string my_robot;
...
my_robot := RobName();
IF my_robot="" THEN
  TPWrite "This task does not control any TCP robot";
ELSE
  TPWrite "This task controls TCP robot with name "+ my_robot;
ENDIF
```

Der Name des TCP-Roboters, der von dieser Programmtask gesteuert wird, wird an das FlexPendant geschrieben. Wenn kein TCP-Roboter gesteuert wird, wird geschrieben, dass die Task keinen Roboter steuert.

Argumente

```
TPWrite String [\Num] | [\Bool] | [\Pos] | [\Orient] | [\Dnum]
```

`String`

Datentyp: `string`

Die Zeichenfolge, die geschrieben werden soll (maximal 80 Zeichen mit 40 Zeichen/Zeile).

`[\Num]`

Numeric

Datentyp: `num`

Die Daten, deren numerischer Wert nach der Zeichenfolge geschrieben werden soll.

`[\Bool]`

Boolean

Datentyp: `bool`

Die Daten, deren logischer Wert nach der Zeichenfolge geschrieben werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.298 TPWrite - Schreiben am FlexPendant
RobotWare - OS
Fortsetzung

`[\Pos]`

Position

Datentyp: `pos`

Die Daten, deren Position nach der Zeichenfolge geschrieben werden soll.

`[\Orient]`

Orientation

Datentyp: `orient`

Die Daten, deren Orientierung nach der Zeichenfolge geschrieben werden soll.

`[\Dnum]`

Numeric

Datentyp: `dnum`

Die Daten, deren numerischer Wert nach der Zeichenfolge geschrieben werden soll.

Programmabarbeitung

Text, der am FlexPendant geschrieben wird, beginnt immer in einer neuen Zeile. Wenn die Anzeige mit Text gefüllt ist (11 Zeilen), wird der Text um jeweils eine Zeile nach oben verschoben.

Wenn eines der Argumente `\Num`, `\Dnum`, `\Bool`, `\Pos` oder `\Orient` verwendet wird, wird sein Wert zuerst in eine Zeichenfolge konvertiert, bevor er der ersten Zeichenfolge hinzugefügt wird. Die Konvertierung des Werts in eine Zeichenfolge findet wie folgt statt:

Argument	Wert	Zeichenfolge
`\Num`	23	"23"
`\Num`	1.141367	"1.14137"
`\Bool`	TRUE	"TRUE"
`\Pos`	[1817.3,905.17,879.11]	"[1817.3,905.17,879.11]"
`\Orient`	[0.96593,0,0.25882,0]	"[0.96593,0,0.25882,0]"
`\Dnum`	4294967295	"4294967295"

Der Wert wird in eine Zeichenfolge mit RAPID-Standardformat konvertiert. Dies bedeutet im Prinzip 6 signifikante Stellen. Wenn der Dezimalteil kleiner als 0,000005 oder größer als 0,999995 ist, wird die Zahl auf eine Ganzzahl gerundet.

Einschränkungen

Die Argumente `\Num`, `\Dnum`, `\Bool`, `\Pos` und `\Orient` schließen einander aus und können daher nicht gemeinsam in derselben Instruktion verwendet werden.

Syntax

```
TPWrite
  [TPText':='] <expression (IN) of string>
  ['\'Num':=' <expression (IN) of num> ]
  | ['\'Bool':=' <expression (IN) of bool> ]
  | ['\'Pos':=' <expression (IN) of pos> ]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
| ['\'Orient':=' <expression (IN) of orient> ]
| ['\'Dnum':=' <expression (IN) of dnum> ]';'
```

Weitere Informationen

Information über	Siehe
Löschen und Lesen des FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.299 TriggC - Kreisförmige Roboterbewegung mit Ereignissen

Verwendung

`TriggC` (*Trigg Circular*) wird verwendet, um Ausgangssignale und/oder Interrupt-Routinen an festen Positionen zu setzen, während sich der Roboter auf einer kreisförmigen Bahn bewegt.

Eines oder mehrere (maximal 25) Ereignisse können mithilfe der Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert werden. Anschließend kann mit der Instruktion `TriggC` auf diese Definitionen verwiesen werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggC`:

Siehe auch *Weitere Beispiele auf Seite 909*.

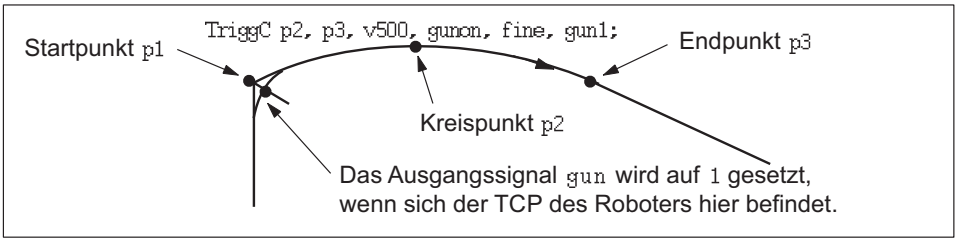
Beispiel 1:

```
VAR triggdata gunon;

TriggIO gunon, 0 \Start \DOp:=gun, 1;
MoveL p1, v500, z50, gun1;
TriggC p2, p3, v500, gunon, fine, gun1;
```

Das digitale Ausgangssignal `gun` wird gesetzt, wenn der TCP die Mitte der Zonenbahn des Punkts `p1` passiert.

Die Abbildung zeigt ein Beispiel für ein positionsabhängiges E/A-Ereignis.



xx0500002267

Argumente

```
TriggC [\Conc] CirPoint ToPoint [\ID] Speed [\T] Trigg_1 |
     TriggArray [\T2] [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone
     [\Inpos] Tool [\WObj] [ \Corr ] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp:`switch`

Folgende Instruktionen werden ausgeführt, während sich der Roboter bewegt. Das Argument wird normalerweise nicht verwendet, kann aber bei der Verwendung von Fly-By-Punkten verwendet werden, um unerwünschte Stopps zu vermeiden,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

die durch eine überlastete CPU ausgelöst werden. Dies ist nützlich, wenn die programmierten Punkte bei hohen Geschwindigkeiten sehr nahe beieinander liegen. Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss. Es kann zudem zur Abstimmung der Abarbeitung mit der Roboterbahn verwendet werden, um die Warnung „50024 Corner path failure“ oder den Fehler „50082 Declaration limit“ zu vermeiden.

Wenn das Argument `\Conc` verwendet wird, ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht erlaubt.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`CirPoint`

Datentyp: `robtarget`

Der Bogenpunkt des Roboters: Eine ausführlichere Beschreibung von kreisförmiger Bewegung finden Sie in der Erläuterung der Instruktion `MoveC`. Der Bogenpunkt wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`ToPoint`

Datentyp: `robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.299 TriggC - Kreisförmige Roboterbewegung mit Ereignissen
RobotWare - OS
Fortsetzung

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Trigg_1`

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`TriggArray`

Trigg Data Array Parameter

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm definiert wurden mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO`.

Die Beschränkung beträgt 25 Elemente im Array, und 1 bis 25 Triggerbedingungen müssen definiert werden.

Es ist nicht möglich, die optionalen Argumente `T2`, `T3`, `T4`, `T5`, `T6`, `T7` oder `T8` zur gleichen Zeit zu verwenden, wenn das `TriggArray`-Argument verwendet wird.

`[ \T2 ]`

Trigg 2

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T3 ]`

Trigg 3

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T4 ]`

Trigg 4

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \T5 ]`

Trigg 5

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheck`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T6 ]`

Trigg 6

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T7 ]`

Trigg 7

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T8 ]`

Trigg 8

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`Zone`

Datentyp: `zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Inpos ]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.299 TriggC - Kreisförmige Roboterbewegung mit Ereignissen
RobotWare - OS
Fortsetzung

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \Corr ]`

Correction

Datentyp: `switch`

Korrekturdaten, die durch die Instruktion `CorrWrite` in einen Korrektureintrag geschrieben wurden, werden der Bahn und der Zielposition hinzugefügt, falls dieses Argument vorhanden ist.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

> **Hinweis**
>
> Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Siehe die Instruktion `MoveC` für weitere Informationen über kreisförmige Bewegung.

Wenn die Triggerbedingungen erfüllt sind, während sich der Roboter dem Endpunkt nähert, werden die definierten Triggeraktivitäten ausgeführt. Die Triggerbedingungen sind entweder einen bestimmten Abstand vor dem Endpunkt der Instruktion oder an einem eingestellten Abstand nach dem Startpunkt der Instruktion oder zu einem bestimmten Zeitpunkt (auf eine kurze Zeit begrenzt) vor dem Endpunkt der Instruktion erfüllt.

Bei der schrittweisen Vorwärtsabarbeitung werden die E/A-Aktivitäten ausgeführt, aber die Interrupt-Routinen laufen nicht ab. Bei der schrittweisen Rückwärtsabarbeitung werden überhaupt keine Triggeraktivitäten ausgeführt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AO_LIM`	Das programmierte `ScaleValue`-Argument für das angegebene analoge Ausgangssignal `AOp` liegt in einigen der verbundenen `TriggSpeed`-Instruktionen zusammen mit dem programmierten `Speed`-Argument in dieser Instruktion außerhalb der Grenzen für das analoge Signal.
`ERR_DIPLAG_LIM`	Das programmierte `DipLag`-Argument in einer verbundenen `TriggSpeed`-Instruktion ist im Verhältnis zur verwendeten Event Preset Time in den Systemparametern zu groß.
`ERR_NORUNUNIT`	Beim Einstieg in die Instruktion kein Kontakt mit dem E/A-Gerät vorhanden ist und die verwendeten triggdata erfordern, dass das E/A-Gerät in Betrieb ist (d. h. dass in den triggdata ein Signal verwendet wird).
`ERR_CONC_MAX`	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument `\Conc` verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TriggC` sind unten aufgeführt.

Beispiel 1:

```
VAR intnum intno1;
VAR triggdata trigg1;
...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.299 TriggC - Kreisförmige Roboterbewegung mit Ereignissen
RobotWare - OS
Fortsetzung

```
PROC main()
  ...
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 0.1 \Time, intno1;
  ...
  TriggC p1, p2, v500, trigg1, fine, gun1;
  TriggC p3, p4, v500, trigg1, fine, gun1;
  ...
  IDelete intno1;
```

Die Interrupt-Routine `trap1` wird ausgeführt, wenn sich der Werkzeugarbeitspunkt an einer Position 0,1 s vor dem Punkt `p2` bzw. `p4` befindet.

Beispiel 2:

```
VAR num Distance:=0;
VAR triggdata trigg_array{25};
VAR signaldo myaliassignaldo;
VAR string signalname;
...
PROC main()
  ...
  FOR i FROM 1 TO 25 DO
    signalname:="do";
    signalname:=signalname+ValToStr(i);
    AliasIO signalname, myaliassignaldo;
    TriggEquip trigg_array{i}, Distance \Start, 0
         \DOp:=myaliassignaldo, SetValue:=1;
    Distance:=Distance+10;
  ENDFOR
  TriggC p1, p2, v500, trigg_array, z30, tool2;
  MoveC p3, p4, v500, z30, tool2;
  ...
```

Die digitalen Ausgangssignale `do1` bis `do25` werden während der Bewegung zu `p2` festgelegt. Der Abstand zwischen den Signaleinstellungen beträgt 10 mm.

Einschränkungen

Allgemeine Einschränkungen gemäß der Instruktion `MoveC`.

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion `TriggC` kürzer als üblich ist, sind eventuell einige oder alle Triggerbedingungen sofort und in derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine „unvollständige Bewegung“ nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

WARNUNG

Die Instruktion `TriggC` sollte nie vom Anfang gestartet werden, wenn der Roboter hinter dem Kreispunkt positioniert ist. Anderenfalls arbeitet der Roboter die programmierte Bahn nicht weiter ab (die Positionierung an der kreisförmigen Bahn erfolgt nicht in der programmierten Richtung).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`TriggC` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggC
  [ '\' Conc ',']
  [ CirPoint' :=' ] < expression (IN) of robtarget > ','
  [ ToPoint' :=' ] < expression (IN) of robtarget > ','
  [ '\' ID ':=' < expression (IN) of identno >] ','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ] ','
  [Trigg_1 ':=' ] < variable (VAR) of triggdata > |
  [TriggArray ':=' ] < array variable {*} (VAR) of triggdata >
  [ '\' T2 ':=' < variable (VAR) of triggdata > ]
  [ '\' T3 ':=' < variable (VAR) of triggdata > ]
  [ '\' T4 ':=' < variable (VAR) of triggdata > ]
  [ '\' T5 ':=' < variable (VAR) of triggdata > ]
  [ '\' T6 ':=' < variable (VAR) of triggdata > ]
  [ '\' T7 ':=' < variable (VAR) of triggdata > ]
  [ '\' T8 ':=' < variable (VAR) of triggdata > ] ','
  [Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos' :=' < expression (IN) of stoppointdata > ]','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj' :=' < persistent (PERS) of wobjdata > ]
  [ '\' Corr ]
  [ '\' TLoad' :=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Lineare Bewegung mit Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*
Achsenbewegung mit Triggern	*TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Kreisförmiges Bewegen des Roboters	*MoveC - Kreisförmige Bewegung des Roboters auf Seite 389*
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung auf Seite 983*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.299 TriggC - Kreisförmige Roboterbewegung mit Ereignissen

RobotWare - OS

Fortsetzung

Information über	Siehe
Handhabung `triggdata`	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen auf Seite 921* *TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable auf Seite 919* *TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist auf Seite 1563*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Kreisbewegung	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.300 TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position

Verwendung

`TriggCheckIO` wird verwendet, um Bedingungen für das Prüfen eines digitalen Eingangs-/Ausgangssignals, einer Gruppe digitaler Eingangs-/Ausgangssignale oder eines analogen Eingangs-/Ausgangssignals an einer festen Position auf der Bewegungsbahn des Roboters zu definieren. Wenn die Bedingung erfüllt ist, erfolgt keine spezifische Aktion. Wenn sie jedoch nicht erfüllt ist, wird eine Interrupt-Routine ausgeführt, nachdem der Roboter optional so schnell wie möglich auf der Bahn gestoppt wurde.

Für eine E/A-Prüfung mit fester Position kompensiert `TriggCheckIO` die Verzögerung in der Steuerung (zwischen Roboter und Servo).

Die definierten Daten können in einer oder mehreren nachfolgenden Instruktionen `TriggL`, `TriggC` oder `TriggJ` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggCheckIO`:

Siehe auch *Weitere Beispiele auf Seite 917*.

Beispiel 1:

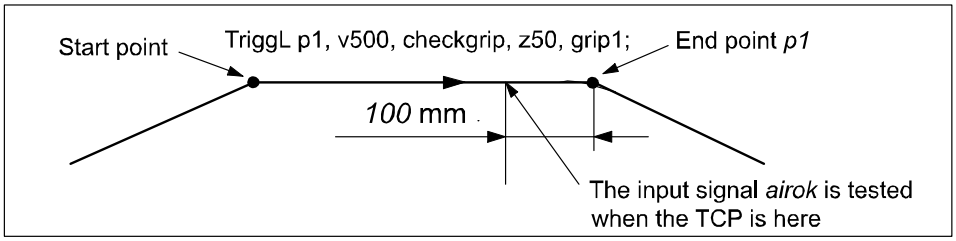
```
VAR triggdata checkgrip;
VAR intnum intno1;

PROC main()
  CONNECT intno1 WITH trap1;
  TriggCheckIO checkgrip, 100, airok, EQ, 1, intno1;

  TriggL p1, v500, checkgrip, z50, grip1;
```

Für das digitale Eingangssignal `airok` wird geprüft, ob es den Wert `1` aufweist, wenn sich der TCP `100` mm vor dem Punkt `p1` befindet. Wenn das Signal gesetzt ist, wird die normale Abarbeitung des Programms fortgesetzt. Wenn es nicht gesetzt ist, wird die Interrupt-Routine `trap1` abgearbeitet.

Die Abbildung zeigt ein Beispiel für eine positionsabhängige E/A-Prüfung.



xx0500002254

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.300 TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position

RobotWare - OS

Fortsetzung

Argumente

```
TriggCheckIO TriggData Distance [\Start] | [\Next] | [\Time] Signal
     Relation CheckValue | CheckDvalue [\StopMove] Interrupt
     [\Inhib] [\Mode]
```

`TriggData`

Datentyp: `triggdata`

Variable zum Speichern der `triggdata` , die von dieser Instruktion zurückgegeben werden. Diese `triggdata` werden dann in der nachfolgenden Instruktion `TriggL`, `TriggC` oder `TriggJ` verwendet.

`Distance`

Datentyp: `num`

Definiert die Position auf der Bahn, wo die E/A-Prüfung stattfinden soll.

Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn (gültig, wenn das Argument `\Start` oder `\Time` nicht gesetzt wurde).

Ausführliche Anleitungen finden Sie unter *Programmabarbeitung auf Seite 915*.

`[ \Start ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für Argument `Distance` ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`[ \Next ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für das Argument `Distance` auf dem Weg zum nächsten programmierten Punkt ist. Wenn die `Distance` länger ist als die Entfernung zum nächsten Feinpunkt, wird das Ereignis am Feinpunkt ausgeführt.

`[ \Time ]`

Datentyp: `switch`

Wird verwendet, wenn der Wert für das Argument `Distance` eine Zeitangabe in Sekunden (positiver Wert) statt eines Abstands darstellt.

Zeitlich fest positionierte E/A ist nur möglich für eine kurze Zeitdauer (< 0,5 s), bevor der Roboter den Endpunkt der Instruktion erreicht. Weitere Einzelheiten erfahren Sie im Abschnitt *Einschränkungen*.

`Signal`

Datentyp: `signalxx`

Der Name des zu prüfenden Signals. Kann ein E/A-Signal eines beliebigen Typs sein.

`Relation`

Datentyp: `opnum`

Definiert, wie der aktuelle Wert des Signals mit dem durch Argument `CheckValue` definierten Wert verglichen werden soll. Unter dem Datentyp `opnum` finden Sie die Liste mit vordefinierten Konstanten, die verwendet werden können.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`CheckValue`

Datentyp: `num`

Wert, mit dem der aktuelle Wert des Eingangs- oder Ausgangssignals verglichen werden soll (innerhalb des zulässigen Bereichs für das aktuelle Signal). Bei einem digitalen Signal muss der Wert eine Ganzzahl sein.

Bei einem digitalen Gruppensignal hängt der zulässige Wert von der Anzahl der Signale in der Gruppe ab. Der Höchstwert beim `CheckValue`-Argument ist 8388608, und das entspricht dem Höchstwert für ein digitales Gruppensignal mit 23 Bit (siehe `num`-Bereiche).

`CheckDvalue`

Datentyp: `dnum`

Wert, mit dem der aktuelle Wert des Eingangs- oder Ausgangssignals verglichen werden soll (innerhalb des zulässigen Bereichs für das aktuelle Signal). Bei einem digitalen Signal muss der Wert eine Ganzzahl sein.

Bei einem digitalen Gruppensignal hängt der zulässige Wert von der Anzahl der Signale in der Gruppe ab. Die Höchstanzahl an Bits für ein digitales Gruppensignal beträgt 32 Bits. Mit einer `dnum`-Variablen kann der Wertebereich 0-4294967295 abgedeckt werden, der dem Wertebereich eines digitalen Signals mit 32 Bits entspricht.

`[ \StopMove ]`

Datentyp: `switch`

Legt fest, dass der Roboter bei nicht erfüllter Bedingung so schnell wie möglich auf der Bahn stoppt, bevor die Interrupt-Routine ausgeführt wird.

`Interrupt`

Datentyp: `intnum`

Variable zur Identifizierung der Interrupt-Routine, die ausgeführt werden soll.

`[ \Inhib ]`

Inhibit

Datentyp: `bool`

Der Name des Flags einer persistenten Variablen zur Verhinderung der Ausführung der Interrupt-Routine.

Wenn das optionale Argument verwendet wird und der aktuelle Wert des angegebenen Flags an der Positionszeit für die E/A-Prüfung TRUE ist, wird die Prüfung nicht durchgeführt.

`[ \Mode ]`

Datentyp: `triggmode`

Wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

Programmabarbeitung

Beim Ausführen der Instruktion `TriggCheckIO` wird die Triggerbedingung in einer angegebenen Variablen für Argument `TriggData` gespeichert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

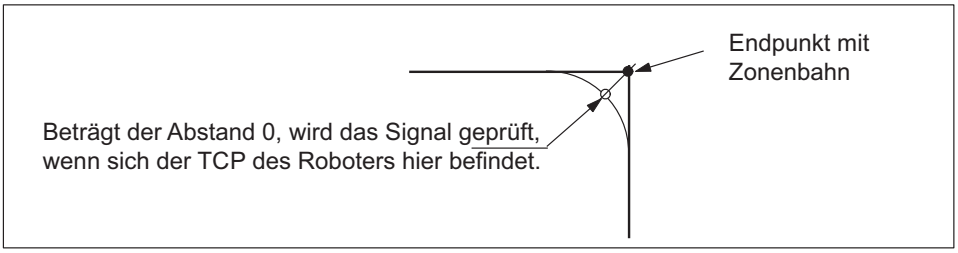
1.300 TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position
RobotWare - OS
Fortsetzung

Wenn danach eine der Instruktionen `TriggL`, `TriggC` oder `TriggJ` abgearbeitet wird, gilt hinsichtlich der Definitionen in `TriggCheckIO` Folgendes:

Die Tabelle beschreibt den im Argument `Distance` angegebenen Abstand:

Lineare Bewegung	Der geradlinige Abstand
Kreisbewegung	Die Länge des Kreisbogens
Nicht lineare Bewegung	Die ungefähre Bogenlänge entlang der Bahn (für angemessene Genauigkeit sollte der Abstand nicht die Hälfte der Bogenlänge überschreiten.)

Die Abbildung zeigt die positionsabhängige E/A-Prüfung auf einer Zonenbahn.



xx0500002256

Die positionsabhängige E/A-Prüfung erfolgt, wenn der Startpunkt (Endpunkt) passiert wird, falls der angegebene Abstand vom Endpunkt (Startpunkt) nicht innerhalb der Bewegungslänge der aktuellen Instruktion liegt (`TriggL...`).

Wenn sich der TCP des Roboters an der angegebenen Position auf der Bahn befindet, führt das System die folgende E/A-Prüfung durch:

- Liest den Wert des E/A-Signals ab.
- Vergleicht den gelesenen Wert mit `CheckValue` gemäß angegebener `Relation`.
- Wenn der Vergleich TRUE ergibt, erfolgt keine weitere Aktion.
- Wenn der Vergleich FALSE ergibt, wird Folgendes ausgeführt:
- Wenn der optionale Parameter `\StopMove` vorhanden ist, wird der Roboter so schnell wie möglich auf der Bahn gestoppt.
- Die angegebene Interrupt-Routine wird generiert und abgearbeitet.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AO_LIM`	Das programmierte `CheckValue`- oder `CheckDvalue`-Argument für das angegebene analoge Ausgangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_GO_LIM`	Das programmierte `CheckValue`- oder `CheckDvalue`-Argument für das angegebene digitale Gruppenausgangssignal `Signal` liegt außerhalb des zulässigen Bereichs.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion TriggCheckIO sind unten aufgeführt.

Beispiel 1:

```
VAR triggdata checkgate;
VAR intnum gateclosed;

PROC main()
  CONNECT gateclosed WITH waitgate;
  TriggCheckIO checkgate,150, gatedi, EQ, 1 \StopMove, gateclosed;
  TriggL p1, v600, checkgate, z50, grip1;
  ...
TRAP waitgate
  ! Block movement
  StopMove;
  ! log some information
  ...
  ! Wait until signal is set
  WaitDI gatedi,1;
  ! Unlock block, and resume movement
  StartMove;
ENDTRAP
```

Es wird geprüft, ob das Gate zur Bedienung des nächsten Werkstücks geöffnet ist (es wird geprüft, ob das digitale Eingang Signal gatedi den Wert 1 hat), wenn das TCP nicht 150 mm vor dem Punkt p1 ist. Wenn es geöffnet ist, bewegt sich der Roboter weiter zu p1 und fährt fort. Wenn es nicht geöffnet ist, wird der Roboter auf der Bahn gestoppt, und die Interrupt Routine waitgate wird ausgeführt. Dieser Interrupt blockiert weitere Bewegungen, protokolliert einige Informationen und wartet in der Regel darauf, dass die Bedingungen OK sind, um eine StartMove Instruktion auszuführen, um den unterbrochenen Pfad neu zu starten.

Einschränkungen

Abstandsabhängige E/A-Prüfungen (ohne das Argument \Time) sind für Fly-By-Punkte (Zonenbahn) gedacht. Abstandsabhängige E/A-Prüfungen, die Stopppunkte verwenden, führen zu schlechterer Genauigkeit als unten angegeben.

Zeitabhängige E/A-Prüfungen (mit dem Argument \Time) sind für Stopppunkte gedacht. Zeitabhängige E/A-Prüfungen, die Fly-By-Punkte verwenden, führen zu schlechterer Genauigkeit als unten angegeben.

Zeitabhängige E/A-Prüfungen können nur ab dem Endpunkt der Bewegung angegeben werden. Diese Zeit darf die aktuelle Bremszeit des Roboters nicht überschreiten, d. h. maximal ca. 0,5 s s (typische Werte bei der Geschwindigkeit 500 mm/s für IRB 2400 150 ms und für IRB 6400 250 ms). Wenn die angegebene

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.300 TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position
RobotWare - OS
Fortsetzung

Zeit die aktuelle Bremszeit überschreitet, wird die E/A-Prüfung dennoch generiert, jedoch erst nach Beginn des Bremsvorgangs (also später als angegeben). Die gesamte Bewegungszeit für die aktuelle Bewegung kann während kurzer und schneller Bewegungen verwendet werden.

Die typische Absolutgenauigkeit für die Prüfung digitaler Eingänge beträgt +/- 5 ms. Die typische Wiederholgenauigkeit für die Prüfung digitaler Eingänge beträgt +/-2 ms.

`TriggCheckIO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggCheckIO
  [ TriggData ':=' ] < variable (VAR) of triggdata> ','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ] | [ '\' Next ] | [ '\' Time ] ','
  [ Signal ':=' ] < variable (VAR) of anytype> ','
  [ Relation' :=' ] < expression (IN) of opnum> ','
  [ CheckValue' :=' ] < expression (IN) of num>
  | [ CheckDvalue' :=' ] < expression (IN) of dnum>
  [ '\' StopMove] ','
  [ Interrupt' :=' ] < variable(VAR) of intnum>
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ '\' Mode' :=' < expression (IN) of triggmode> ] ';'
```

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von positions-/zeitabhängigem E/A-Ereignis	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923*
Definition von positionsbezogenen Interrupts	*TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Speichern von Triggerdaten	*triggdata – Positioniereereignisse, trigg auf Seite 1833*
Definieren von verschiedenen Trigger-Aktionsmodi	*triggmode - Triggeraktionsmodus auf Seite 1840*
Definition von Vergleichsoperatoren	*opnum - Vergleichsoperator auf Seite 1743*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.301 TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable

Verwendung

`TriggDataCopy` dient dazu, den Inhalt in eine `triggdata`-Variable zu kopieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggDataCopy`.

Beispiel 1:

```
VAR triggdata trigg_array{25};
...
PROC MyTriggProcL(robtarget myrobt, \VAR triggdata T1 \VAR triggdata
     T2 \VAR triggdata T3)
  VAR num triggcnt:=2;
  ! Reset entire trigg_array array before using it
  FOR i FROM 1 TO 25 DO
    TriggDataReset trigg_array{i};
  ENDFOR
  TriggEquip trigg_array{1}, 10 \Start, 0 \DOp:=do1, SetValue:=1;
  TriggEquip trigg_array{2}, 40 \Start, 0 \DOp:=do2, SetValue:=1;
  ! Check if optional argument is present,
  ! and if any trigger condition has been setup in T1
  IF Present(T1) AND TriggDataValid(T1) THEN
    ! Copy actual trigger condition to trigg_array
    TriggDataCopy T1, trigg_array{triggcnt};
    Incr triggcnt;
  ENDIF
  IF Present(T2) AND TriggDataValid(T2) THEN
    Incr triggcnt;
    TriggDataCopy T2, trigg_array{triggcnt};
  ENDIF
  IF Present(T3) AND TriggDataValid(T3) THEN
    Incr triggcnt;
    TriggDataCopy T3, trigg_array{triggcnt};
  ENDIF
  TriggL p1, v500, trigg_array, z30, tool2;
  ...
```

Das Verfahren `MyTriggProcL` oben nutzt die `TriggDataCopy`-Instruktion, um die optionalen `triggdata`-Argument an die richtige Stelle im `triggdata`-Array zu kopieren, das in der `TriggL`-Instruktion verwendet wird.

Argumente

```
TriggDataCopy Source Destination
```

`Source`

Datentyp: `triggdata`

Die `triggdata`-Variable, aus der kopiert werden soll.

`Destination`

Datentyp: `triggdata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.301 TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable

RobotWare - OS

Fortsetzung

Die `triggdata`-Variable, in die kopiert werden soll.

Programmabarbeitung

Die `TriggDataCopy`-Instruktion dient dazu, Daten von einer `triggdata`-Variablen in eine andere `triggdata`-Variable zu kopieren. Diese Instruktion kann bei der Arbeit mit `triggdata`-Array-Variablen hilfreich sein.

Einschränkungen

`TriggDataCopy` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggDataCopy
  [Source ':=' ] < variable (VAR) of triggdata > ','
  [Destination ':=' ] < variable (VAR) of triggdata > ';'
```

Weitere Informationen

Information über	Siehe
Lineare Bewegung mit Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*
Achsenbewegung mit Triggern	*TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Kreisförmige Bewegung mit Triggern	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung auf Seite 983*
Handhabung `triggdata`	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen auf Seite 921* *TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist auf Seite 1563*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.302 TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen

Verwendung

`TriggDataReset` dient dazu, den Inhalt in einer `triggdata`-Variablen zurückzusetzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggDataReset`.

Beispiel 1:

```
VAR triggdata trigg_array{25};
...
PROC MyTriggProcL(robtarget myrobt, \VAR triggdata T1 \VAR triggdata
     T2 \VAR triggdata T3)
  VAR num triggcnt:=2;
  ! Reset entire trigg_array array before using it
  FOR i FROM 1 TO 25 DO
    TriggDataReset trigg_array{i};
  ENDFOR
  TriggEquip trigg_array{1}, 10 \Start, 0 \DOp:=do1, SetValue:=1;
  TriggEquip trigg_array{2}, 40 \Start, 0 \DOp:=do2, SetValue:=1;
  ! Check if optional argument is present,
  ! and if any trigger condition has been setup in T1
  IF Present(T1) AND TriggDataValid(T1) THEN
    ! Copy actual trigger condition to trigg_array
    TriggDataCopy trigg_array{triggcnt}, T1;
    Incr triggcnt;
  ENDIF
  IF Present(T2) AND TriggDataValid(T2) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T2;
  ENDIF
  IF Present(T3) AND TriggDataValid(T3) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T3;
  ENDIF
  TriggL p1, v500, trigg_array, z30, tool2;
  ...
```

Das Verfahren `MyTriggProcL` oben nutzt die `TriggDataReset`-Instruktion zum Zurücksetzen des `triggdata`-Arrays, bevor es verwendet wird.

Argumente

```
TriggDataReset TriggData
```

`TriggData`

Datentyp: `triggdata`

Die `triggdata`-Variable, die zurückgesetzt werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.302 TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen

RobotWare - OS

Fortsetzung

Programmabarbeitung

Die `TriggDataReset`-Instruktion dient dazu, alle Triggerbedingungen zu entfernen, die zuvor in einer `triggdata`-Variablen verwendet wurden. Diese Instruktion kann bei der Arbeit mit `triggdata`-Array-Variablen hilfreich sein.

Einschränkungen

`TriggDataReset` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggDataReset
  [TriggData ':=' ] < variable (VAR) of triggdata > ';'
```

Weitere Informationen

Information über	Siehe
Lineare Bewegung mit Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*
Achsenbewegung mit Triggern	*TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Kreisförmige Bewegung mit Triggern	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung auf Seite 983*
Handhabung `triggdata`	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable auf Seite 919* *TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist auf Seite 1563*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.303 TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn

Verwendung

`TriggEquip` (*Trigg Equipment*) wird verwendet, um Bedingungen und Aktionen für das Setzen eines digitalen Ausgangssignals, einer Gruppe digitaler Ausgangssignale oder eines analogen Ausgangssignals an einer festen Position auf der Bewegungsbahn des Roboters zu definieren. Dabei kann die Verzögerung von externen Einrichtungen kompensiert werden.

`TriggIO` (nicht `TriggEquip`) sollte immer verwendet werden, wenn in der Nähe eines Stopppunkts eine hohe Genauigkeit der E/A-Einstellungen erforderlich ist.

Die definierten Daten können in einer oder mehreren nachfolgenden Instruktionen `TriggL`, `TriggC` oder `TriggJ` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggEquip`:

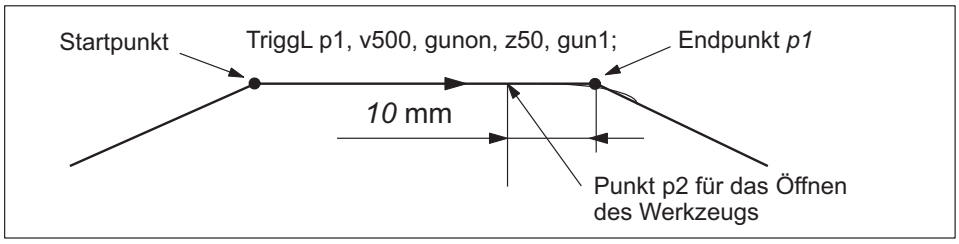
Siehe auch *Weitere Beispiele auf Seite 927*.

Beispiel 1:

```
VAR triggdata gunon;
...
TriggEquip gunon, 10, 0.1 \DOp:=gun, 1;
TriggL p1, v500, gunon, z50, gun1;
```

Das Werkzeug `gun1` beginnt sich zu öffnen, wenn sich sein TCP `0,1` s vor dem fiktiven Punkt `p2` (`10` mm vor Punkt `p1`) befindet. Das Werkzeug ist vollständig geöffnet, wenn der TCP Punkt `p2` erreicht.

Die Abbildung zeigt ein Beispiel für ein positions- und zeitabhängiges E/A-Ereignis.



xx0500002260

Argumente

```
TriggEquip TriggData Distance [\Start] | [\Next] EquipLag [\DOp]
     | [\GOp]| [\AOp] | [\ProcID] SetValue | SetDvalue [\Inhib]
     [\InhibSetValue] [\Mode]
```

`TriggData`

Datentyp: `triggdata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Variable zum Speichern der `triggdata` , die von dieser Instruktion zurückgegeben werden. Diese `triggdata` werden dann in der nachfolgenden Instruktion `TriggL`, `TriggC` oder `TriggJ` verwendet.

`Distance`

Datentyp: `num`

Definiert die Position auf der Bahn, wo das E/A-Ereignis stattfinden soll.

Angegeben als die Entfernung in mm (positiver Wert) vom Endpunkt der Bewegungsbahn in Richtung Startpunkt (gültig, wenn die Argumente `\Start` und `\Next` nicht festgelegt wurden).

Ausführliche Anleitungen finden Sie unter *Programmabarbeitung auf Seite 926*.

`[ \Start ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für Argument `Distance` ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`[ \Next ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für das Argument `Distance` auf dem Weg zum nächsten programmierten Punkt ist. Wenn die `Distance` länger ist als die Entfernung zum nächsten Feinpunkt, wird das Ereignis am Feinpunkt ausgeführt.

`EquipLag`

Equipment Lag

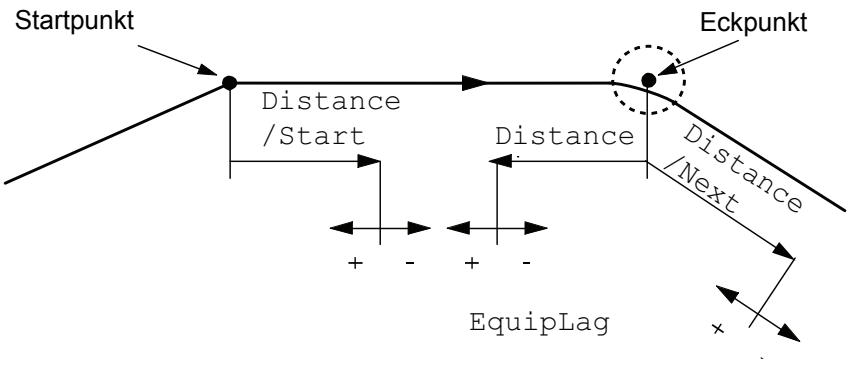
Datentyp: `num`

Gibt die Verzögerung der externen Einrichtung in s an.

Verwenden Sie zur Kompensierung der Verzögerung einen positiven Argumentwert. Ein positiver Argumentwert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, bevor der TCP physisch den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung erreicht hat.

Ein negativer Argumentwert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, nachdem der TCP den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung physisch passiert hat.

Die Abbildung zeigt die Verwendung von Argument `EquipLag`.



xx0500002262

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \DOp ]`

Digital Output

Datentyp: `signaldo`

Der Name des Signals, wenn ein digitales Ausgangssignal geändert werden soll.

`[ \GOp ]`

Group Output

Datentyp: `signalgo`

Der Name des Signals, wenn eine Gruppe von digitalen Ausgangssignalen geändert werden soll.

`[ \AOp ]`

Analog Output

Datentyp: `signalao`

Der Name des Signals, wenn ein analoges Ausgangssignal geändert werden soll.

`[ \ProcID ]`

Process Identity

Datentyp: `num`

Nicht für den Einsatz durch Endanwender implementiert.

(Die Identität des IPM-Prozesses, der das Ereignis empfangen soll. Der Selektor wird im Argument `SetValue` angegeben.)

`SetValue`

Datentyp: `num`

Der gewünschte Wert des Signals (innerhalb des zulässigen Bereichs für das aktuelle Signal). Wenn das Signal ein digitales Signal ist, muss es ein ganzzahliger Wert sein. Wenn das Signal ein digitales Gruppensignal ist, hängt der zulässige Wert von der Anzahl der Signale in der Gruppe ab. Der maximale Wert, der im `SetValue`-Argument verwendet werden kann, ist 8388608, und dies ist der maximal zulässige Wert für ein digitales Gruppensignal von 23 Bit (siehe Wertebereiche für `num`).

`SetDvalue`

Datentyp: `dnum`

Gewünschter Wert des Signals (innerhalb des zulässigen Bereichs für das aktuelle Signal). Bei einem digitalen Signal muss der Wert eine Ganzzahl sein. Bei einem digitalen Gruppensignal hängt der zulässige Wert von der Anzahl der Signale in der Gruppe ab. Die Höchstanzahl an Bits für ein digitales Gruppensignal beträgt 32 Bits. Mit einer `dnum`-Variablen kann der Wertebereich 0-4294967295 abgedeckt werden, der dem Wertebereich eines digitalen Signals mit 32 Bits entspricht.

`[ \Inhib ]`

Inhibit

Datentyp: `bool`

Der Name eines Flags für persistente Variablen, das das Setzen des Signals zur Laufzeit verhindert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn das optionale Argument verwendet wird und der aktuelle Wert des angegebenen Flags zu dem Zeitpunkt, an dem das Signal gesetzt werden soll, TRUE ist, wird das angegebene Signal (`DOp`, `GOp` oder `AOp`) nicht auf den angegebenen Wert, sondern auf 0 gesetzt.

`[ \InhibSetValue ]`

InhibitSetValue

Datentyp: `bool, num or dnum`

Der Name einer persistenten Variablen des Datentyps `bool`, `num` oder `dnum` oder jedes Alias dieser drei Basisdatentypen.

Das optionale Argument kann nur gemeinsam mit dem optionalen Argument `Inhib` verwendet werden.

Wenn dieses optionale Argument verwendet wird, und der Wert des Flags der persistenten Variablen, die im optionalen Argument `Inhib` verwendet wird, an der Positionszeit für die Signaleinstellung TRUE ist, wird der Wert der persistenten Variable gelesen, die im optionalen Argument `InhibSetValue` verwendet wird. Dieser Wert wird zur Einstellung von `DOp`, `GOp` oder des Signals `AOp` verwendet.

Bei Verwendung einer persistenten booleschen Variable wird der Wert TRUE in den Wert 1 und FALSE in den Wert 0 übertragen.

`[ \Mode ]`

Datentyp: `triggmode`

Wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

Programmabarbeitung

Beim Ausführen der Instruktion `TriggEquip` wird die Triggerbedingung in der angegebenen Variablen für Argument `TriggData` gespeichert.

Wenn danach eine der Instruktionen `TriggL`, `TriggC` oder `TriggJ` abgearbeitet wird, gilt hinsichtlich der Definitionen in `TriggEquip` Folgendes:

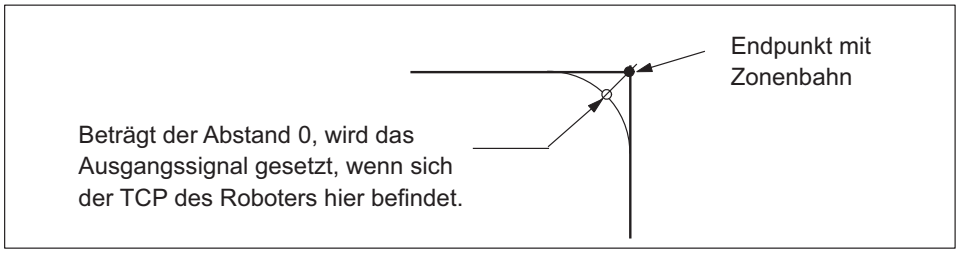
Die Tabelle beschreibt den im Argument `Distance` angegebenen Abstand:

Lineare Bewegung	Der geradlinige Abstand
Kreisbewegung	Die Länge des Kreisbogens
Nicht lineare Bewegung	Die ungefähre Bogenlänge entlang der Bahn (für angemessene Genauigkeit sollte der Abstand nicht die Hälfte der Bogenlänge überschreiten.)

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Abbildung zeigt die positions- und zeitabhängige E/A-Prüfung auf einer Zonenbahn.



xx0500002263

Das positions- und zeitabhängige Ereignis wird generiert, wenn der Startpunkt (Endpunkt) passiert wird, falls der angegebene Abstand vom Endpunkt (Startpunkt) nicht innerhalb der Bewegungslänge der aktuellen Instruktion liegt (`TriggL...`). Falls das Argument `EquipLag` eine negative Zeit (Verzögerung) enthält, kann das E/A-Signal nach dem Endpunkt gesetzt werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AO_LIM`	Das programmierte `SetValue`-Argument für das angegebene analoge Ausgangssignal `AOp` liegt außerhalb des zulässigen Bereichs.
`ERR_GO_LIM`	Das programmierte `SetValue`- oder `SetDvalue`-Argument für das angegebene digitale Gruppenausgangssignal `GOp` liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TriggEquip` sind unten aufgeführt.

Beispiel 1:

```
VAR triggdata glueflow;
...
TriggEquip glueflow, 1 \Start, 0.05 \AOp:=glue, 5.3;
MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, glueflow, z50, tool1;
```

Das analoge Ausgangssignal `glue` wird auf den Wert `5.3` gesetzt, wenn der TCP einen Punkt passiert, der sich `1` mm hinter dem Startpunkt `p1` befindet, und die Verzögerung der Einrichtung wird mit `0.05` s kompensiert.

Beispiel 2:

```
...
TriggL p3, v500, glueflow, z50, tool1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.303 TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn
RobotWare - OS
Fortsetzung

Das analoge Ausgangssignal `glue` wird erneut auf den Wert `5.3` gesetzt, wenn der TCP einen Punkt passiert, der sich `1` mm hinter dem Startpunkt `p2` befindet.

Einschränkungen

Abstandsabhängige E/A-Ereignisse sind für Fly-By-Punkte (Zonenbahn) gedacht. Die Verwendung von Stopppunkten führt zu einer geringeren Genauigkeit als unten angegeben.

Hinsichtlich der Genauigkeit für E/A-Ereignisse mit Abstand und Fly-By-Punkten (Bahnverlauf) gilt Folgendes, wenn ein digitaler Ausgang im angegebenen Abstand vom Start- oder Endpunkt in der Instruktion `TriggL` oder `TriggC` gesetzt wird:

- Die unten angegebene Genauigkeit ist für einen positiven `EquipLag` von weniger als 40 ms gültig, entsprechend der Verzögerung im Roboterservo, ohne Ändern des Systemparameters *Event Preset Time*. Die Verzögerung kann je nach Robotertyp variieren.
- Die nachfolgend angegebene Genauigkeit gilt bei Verwendung eines positiven `EquipLag`, der weniger als den konfigurierten *Event Preset Time* in den Systemparametern entspricht.
- Die unten angegebene Genauigkeit gilt nicht für einen positiven `EquipLag`, der größer als der in den Systemparametern konfigurierte *Event Preset Time* ist. In diesem Fall wird eine Näherungsmethode verwendet, bei der die dynamischen Grenzen des Roboters nicht berücksichtigt werden. Dann muss `SingArea \Wrist` verwendet werden, um eine akzeptable Genauigkeit zu erzielen.
- Die nachstehend angegebene Geschwindigkeit gilt bei Verwendung eines negativen `EquipLag`.

Die typischen absoluten Genauigkeitswerte zur Einstellung digitaler Ausgänge ist: ±5 ms.

Die typischen wiederholten Genauigkeitswerte zur Einstellung digitaler Ausgänge ist: ±2 ms.

`TriggEquip` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggEquip
  [ TriggData ':=' ] < variable (VAR) of triggdata> ','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ] | [ '\' Next ] ','
  [ EquipLag' :=' ] < expression (IN) of num>
  [ '\' DOp' :=' < variable (VAR) of signaldo> ]
  | [ '\' GOp' :=' < variable (VAR) of signalgo> ]
  | [ '\' AOp' :=' < variable (VAR) of signalao> ]
  | [ '\' ProcID' :=' < expression (IN) of num> ] ','
  [ SetValue' :=' ] < expression (IN) of num>
  | [ SetDvalue' :=' ] < expression (IN) of dnum> ','
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.303 TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn

RobotWare - OS

Fortsetzung

```
[ '\' Mode' :=' < expression (IN) of triggmode> ] ';'
```

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von anderen Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Definieren einer positionsabhängigen E/A-Prüfung	*TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913*
Speichern von Triggerdaten	*triggdata – Positioniereereignisse, trigg auf Seite 1833*
Definieren von verschiedenen Trigger-Aktionsmodi	*triggmode - Triggeraktionsmodus auf Seite 1840*
Setzen von E/A	*SetDO - Ändern des Werts eines digitalen Ausgangssignals auf Seite 710* *SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen auf Seite 713* *SetAO - Ändern des Werts eines analogen Ausgangssignals auf Seite 700*
Konfiguration von Event preset time	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.304 TriggInt - Definieren eines positionsabhängigen Interrupts

Verwendung

`TriggInt` wird verwendet, um Bedingungen und Aktionen für das Ausführen einer Interrupt-Routine an einer angegebenen Position der Roboterbewegungsbahn auszuführen.

Die definierten Daten können in einer oder mehreren nachfolgenden Instruktionen `TriggL`, `TriggC` oder `TriggJ` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` bzw. in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

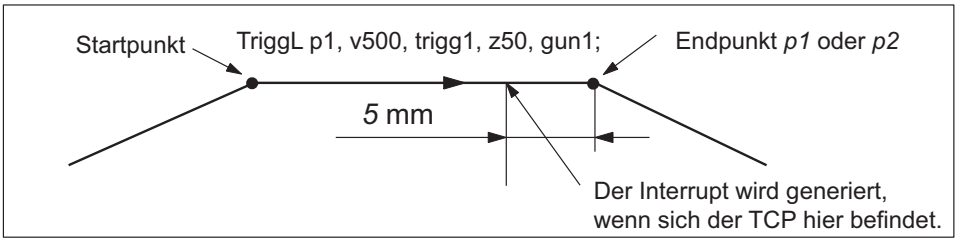
Das folgende Beispiel veranschaulicht die Instruktion `TriggInt`:

Beispiel 1:

```
VAR intnum intno1;
VAR triggdata trigg1;
...
PROC main()
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 5, intno1;
  ...
  TriggL p1, v500, trigg1, z50, gun1;
  TriggL p2, v500, trigg1, z50, gun1;
  ...
  IDelete intno1;
```

Die Interrupt-Routine `trap1` wird ausgeführt, wenn sich der TCP `5` mm vor Punkt `p1` oder `p2` befindet.

Die Abbildung zeigt ein Beispiel eines positionsabhängigen Interrupts.



xx0500002251

Argumente

```
TriggInt TriggData Distance [\Start] | [\Next] | [\Time] Interrupt
    [\Inhib] [\Mode]
```

`TriggData`

Datentyp: `triggdata`

Variable zum Speichern der `triggdata` , die von dieser Instruktion zurückgegeben werden. Diese `triggdata` werden dann in der nachfolgenden Instruktion `TriggL`, `TriggC` oder `TriggJ` verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Distance`

Datentyp: `num`

Definiert die Position auf der Bahn, an der der Interrupt generiert werden soll.

Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn (gültig, wenn das Argument `\Start` oder `\Time` nicht gesetzt wurde).

Ausführliche Anleitungen finden Sie unter *Programmabarbeitung auf Seite 932*.

`[ \Start ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für Argument `Distance` ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`[ \Next ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für das Argument `Distance` auf dem Weg zum nächsten programmierten Punkt ist. Wenn die `Distance` länger ist als die Entfernung zum nächsten Feinpunkt, wird das Ereignis am Feinpunkt ausgeführt.

`[ \Time ]`

Datentyp: `switch`

Wird verwendet, wenn der Wert für das Argument `Distance` eine Zeitangabe in Sekunden (positiver Wert) statt eines Abstands darstellt.

Zeit- und positionsabhängige Interrupts sind nur möglich für eine kurze Zeitdauer (< 0,5 s), bevor der Roboter den Endpunkt der Instruktion erreicht. Weitere Informationen finden Sie im Abschnitt *Einschränkungen*.

`Interrupt`

Datentyp: `intnum`

Variable zur Identifizierung eines Interrupts.

`[ \Inhib ]`

Inhibit

Datentyp: `bool`

Der Name des Flags einer persistenten Variablen zur Verhinderung der Ausführung der Interrupt-Routine.

Wenn das optionale Argument verwendet wird und der aktuelle Wert des angegebenen Flags an der Positionszeit der Interrupt-Ausführung TRUE ist, wird der Interrupt nicht durchgeführt.

`[ \Mode ]`

Datentyp: `triggmode`

Wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.304 TriggInt - Definieren eines positionsabhängigen Interrupts
RobotWare - OS
Fortsetzung

Programmabarbeitung

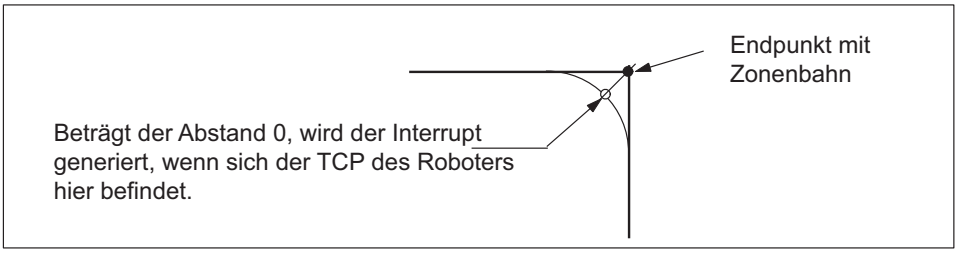
Beim Ausführen der Instruktion `TriggInt` werden Daten in einer angegebenen Variablen für Argument `TriggData` gespeichert und der mit dem Argument `Interrupt` angegebene Interrupt wird aktiviert.

Wenn danach eine der Instruktionen `TriggL`, `TriggC` oder `TriggJ` abgearbeitet wird, gilt hinsichtlich der Definitionen in `TriggInt` Folgendes:

Die Tabelle beschreibt den im Argument `Distance` angegebenen Abstand:

Lineare Bewegung	Der geradlinige Abstand
Kreisbewegung	Die Länge des Kreisbogens
Nicht lineare Bewegung	Die ungefähre Bogenlänge entlang der Bahn (für angemessene Genauigkeit sollte der Abstand nicht die Hälfte der Bogenlänge überschreiten.)

Die Abbildung zeigt einen positionsabhängigen Interrupt auf einer Zonenbahn.



xx0500002253

Der positions- und zeitabhängige Interrupt wird generiert, wenn der Startpunkt (Endpunkt) passiert wird und der angegebene Abstand vom Endpunkt (Startpunkt) nicht innerhalb der Bewegungslänge der aktuellen Instruktion (`TriggL...`) liegt.

Der Interrupt wird als sicherer Interrupt verstanden. Ein sicherer Interrupt kann mit der Instruktion `ISleep` nicht vorübergehend deaktiviert werden. Das Ereignis des sicheren Interrupts wird bei einem Programmstopp und der schrittweisen Abarbeitung in die Warteschlange gestellt. Wenn die kontinuierliche Abarbeitung fortgesetzt wird, wird der Interrupt ausgeführt. Ein sicherer Interrupt wird nur dann verworfen, wenn die Interrupt-Warteschlange voll ist. Dann wird ein Fehler gemeldet. Der Interrupt bleibt nach einem Programm-Reset (z. B. PZ auf main) nicht erhalten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TriggInt` sind unten aufgeführt.

Beispiel 1:

Dieses Beispiel beschreibt die Programmierung der Instruktionen, die zur Generierung von positionsabhängigen Interrupts interagieren.

```
VAR intnum intno2;
VAR triggdata trigg2;
```

- Deklaration der Variablen `intno2` und `trigg2 (` darf nicht aktiviert sein).

```
CONNECT intno2 WITH trap2;
```

- Zuweisung von Interrupt-Nummern, die in der Variablen `intno2` gespeichert werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Die Interrupt-Nummer wird mit der Interrupt-Routine `trap2` verbunden.

```
TriggInt trigg2, 0, intno2;
```

- Die Interrupt-Nummer in der Variablen `intno2` wird als benutzt gekennzeichnet.
- Der Interrupt wird aktiviert.
- Definierte Triggerbedingungen und Interrupt-Nummern werden in der Variablen `trigg2` gespeichert.

```
TriggL p1, v500, trigg2, z50, gun1;
```

- Der Roboter wird an den Punkt `p1` bewegt.
- Wenn der TCP den Punkt `p1` erreicht, wird ein Interrupt generiert und die Interrupt-Routine `trap2` wird ausgeführt.

```
TriggL p2, v500, trigg2, z50, gun1;
```

- Der Roboter wird an den Punkt `p2` bewegt.
- Wenn der TCP den Punkt `p2` erreicht, wird ein Interrupt generiert und die Interrupt-Routine `trap2` wird erneut ausgeführt.

```
IDelete intno2;
```

- Die Interrupt-Nummer in der Variablen `intno2` wird freigegeben.

Einschränkungen

Abstandsabhängige Interrupt-Ereignisse (ohne das Argument `\Time`) sind für so genannte Fly-By-Punkte (Zonenbahn) vorgesehen. Abstandsabhängige Interrupt-Ereignisse, die Stopppunkte verwenden, führen zu geringerer Genauigkeit als unten angegeben.

Zeitabhängige Interrupt Ereignisse (mit dem Argument `\Time`) sind für Stopppunkte vorgesehen. Zeitabhängige Interrupt Ereignisse, die Fly-By-Punkte verwenden, führen zu schlechterer Genauigkeit als unten angegeben. Zeitabhängige E/A-Ereignisse können nur ab dem Endpunkt der Bewegung angegeben werden. Diese Zeit darf die aktuelle Bremszeit des Roboters nicht überschreiten, d. h. maximal ca. 0,5 s (typische Werte bei der Geschwindigkeit 500 mm/s für IRB 2400 150 ms und für IRB 6400 250 ms). Wenn die angegebene Zeit die aktuelle Bremszeit überschreitet, wird das E/A-Ereignis dennoch generiert, jedoch erst nach Beginn des Bremsvorgangs (also später als angegeben). Die gesamte Bewegungszeit für die aktuelle Bewegung kann während kurzer und schneller Bewegungen verwendet werden.

Die typische Absolutgenauigkeit für die Generierung von Interrupts beträgt +/-5 ms. Die typische Wiederholgenauigkeit für die Generierung von Interrupts beträgt +/- 2 ms. Normalerweise entsteht eine Verzögerung von 2 bis 30 ms zwischen Generierung des Interrupts und Reaktion, je nachdem, welche Bewegung zur Zeit des Interrupts ausgeführt wird. Siehe *Technisches Referenzhandbuch - RAPID Overview*.

Für größte Genauigkeit beim Setzen eines Ausgangs an einer festen Position auf der Roboterbahn verwenden Sie die Instruktionen `TriggIO` oder `TriggEquip` anstelle von `TriggInt` mit `SetDO/SetGO/SetAO` in einer Interrupt-Routine.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.304 TriggInt - Definieren eines positionsabhängigen Interrupts

RobotWare - OS

Fortsetzung

`TriggInt` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggInt
  [TriggData ':='] <variable (VAR) of triggdata>','
  [Distance' :=' ] <expression (IN) of num>
  ['\' Start] | ['\' Next ] | ['\' Time]','
  [Interrupt ':='] <variable (VAR) of intnum>
  ['\' Inhib ':=' <persistent (PERS) of bool>]
  ['\' Mode ':=' <expression (IN) of triggmode>]';'
```

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von positionsabhängigem E/A-Ereignis	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923*
Definieren einer positionsabhängigen E/A-Prüfung	*TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913*
Speichern von Triggerdaten	*triggdata – Positioniereregnisse, trigg auf Seite 1833*
Definieren von verschiedenen Trigger-Aktionsmodi	*triggmode - Triggeraktionsmodus auf Seite 1840*
Interrupts	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.305 TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt

Verwendung

`TriggIO` wird verwendet, um Bedingungen und Aktionen für das Setzen eines digitalen Ausgangssignals, einer Gruppe digitaler Ausgangssignale oder eines analogen Ausgangssignals an einer festen Position auf der Bewegungsbahn des Roboters zu definieren.

`TriggIO` (nicht `TriggEquip`) sollte immer verwendet werden, wenn in der Nähe eines Stopppunkts eine hohe Genauigkeit der E/A-Einstellungen erforderlich ist.

Für ein E/A-Ereignis mit fester Position kompensiert `TriggIO` die Verzögerung in der Steuerung (zwischen Roboter und Servo), nicht aber Verzögerungen in externen Einrichtungen. Verwenden Sie `TriggEquip` zur Kompensierung von beiden Verzögerungen.

Die definierten Daten können in einer oder mehreren nachfolgenden Instruktionen `TriggL`, `TriggC` oder `TriggJ` verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggIO`:

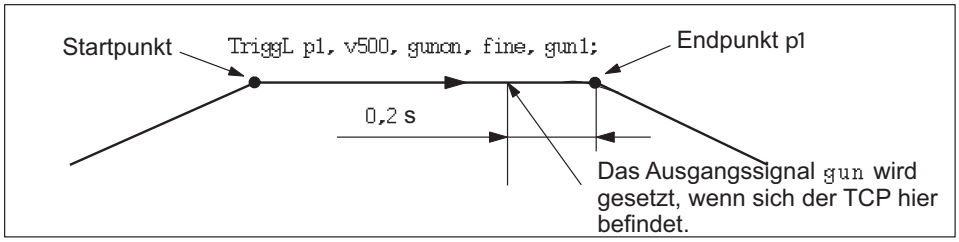
Siehe auch *Weitere Beispiele auf Seite 939*.

Beispiel 1:

```
VAR triggdata gunon;
...
TriggIO gunon, 0.2\Time\DOp:=gun, 1;
TriggL p1, v500, gunon, fine, gun1;
```

Das digitale Ausgangssignal `gun` wird auf den Wert `1` gesetzt, wenn sich der TCP `0,2` Sekunden vor dem Punkt `p1` befindet.

Die Abbildung zeigt ein Beispiel für ein positionsabhängiges E/A-Ereignis.



xx0500002247

Argumente

```
TriggIO TriggData Distance [\Start] | [\Time] [\DOp] | [\GOp]|
     [\AOp] | [\ProcID] SetValue | SetDvalue [\DODelay] [\Inhib]
     [\InhibSetValue] [\Mode]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.305 TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt
RobotWare - OS
Fortsetzung

`TriggData`

Datentyp: `triggdata`

Variable zum Speichern der `triggdata`, die von dieser Instruktion zurückgegeben werden. Diese `triggdata` werden dann in der nachfolgenden Instruktion `TriggL`, `TriggC` oder `TriggJ` verwendet.

`Distance`

Datentyp: `num`

Definiert die Position auf der Bahn, wo das E/A-Ereignis stattfinden soll.

Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn (gültig, wenn das Argument `\Start` oder `\Time` nicht gesetzt wurde).

Weitere Einzelheiten finden Sie in den Abschnitten *Programmabarbeitung auf Seite 938* und *Einschränkungen auf Seite 939*.

`[ \Start ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für Argument `Distance` ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`[ \Time ]`

Datentyp: `switch`

Wird verwendet, wenn der Wert für das Argument `Distance` eine Zeitangabe in Sekunden (positiver Wert) statt eines Abstands darstellt.

Zeitlich fest positionierte E/A ist nur möglich für eine kurze Zeitdauer (< 0,5 s), bevor der Roboter den Endpunkt der Instruktion erreicht. Weitere Einzelheiten erfahren Sie im Abschnitt „Einschränkungen“.

`[ \DOp ]`

Digital Output

Datentyp: `signaldo`

Der Name des Signals, wenn ein digitales Ausgangssignal geändert werden soll.

`[ \GOp ]`

Group Output

Datentyp: `signalgo`

Der Name des Signals, wenn eine Gruppe von digitalen Ausgangssignalen geändert werden soll.

`[ \AOp ]`

Analog Output

Datentyp: `signalao`

Der Name des Signals, wenn ein analoges Ausgangssignal geändert werden soll.

`[ \ProcID ]`

Process Identity

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Nicht für den Einsatz durch Endanwender implementiert.

(Die Identität des IPM-Prozesses, der das Ereignis empfangen soll. Der Selektor wird im Argument `SetValue` angegeben.)

`SetValue`

Datentyp: `num`

Der gewünschte Wert des Signals (innerhalb des zulässigen Bereichs für das aktuelle Signal). Wenn das Signal ein digitales Signal ist, muss es ein ganzzahliger Wert sein. Wenn das Signal ein digitales Gruppensignal ist, hängt der zulässige Wert von der Anzahl der Signale in der Gruppe ab. Der maximale Wert, der im `SetValue`-Argument verwendet werden kann, ist 8388608, und dies ist der maximal zulässige Wert für ein digitales Gruppensignal von 23 Bit (siehe Wertebereiche für `num`).

`SetDvalue`

Datentyp: `dnum`

Gewünschter Wert des Signals (innerhalb des zulässigen Bereichs für das aktuelle Signal). Bei einem digitalen Signal muss der Wert eine Ganzzahl sein. Bei einem digitalen Gruppensignal hängt der zulässige Wert von der Anzahl der Signale in der Gruppe ab. Die Höchstanzahl an Bits für ein digitales Gruppensignal beträgt 32 Bits. Mit einer `dnum`-Variablen kann der Wertebereich 0-4294967295 abgedeckt werden, der dem Wertebereich eines digitalen Signals mit 32 Bits entspricht.

`[ \DODelay ]`

Digital Output Delay

Datentyp: `num`

Zeitverzögerung in Sekunden (positiver Wert) für ein digitales Ausgangssignal, eine Gruppe von Ausgangssignalen oder ein analoges Ausgangssignal.

Wird verwendet, um das Setzen von Ausgangssignalen zu verzögern, nachdem der Roboter die angegebene Position erreicht hat. Es gibt keine Verzögerung, wenn dieses Argument weggelassen wird.

Die Verzögerung wird nicht mit der Bewegung synchronisiert.

`[ \Inhib ]`

Inhibit

Datentyp: `bool`

Der Name eines Flags für persistente Variablen, das das Setzen des Signals zur Laufzeit verhindert.

Wenn das optionale Argument verwendet wird und der aktuelle Wert des angegebenen Flags zu dem Zeitpunkt, an dem das Signal gesetzt werden soll, TRUE ist, wird das angegebene Signal (`DOp`, `GOp` oder `AOp`) nicht auf den angegebenen Wert, sondern auf 0 gesetzt.

`[ \InhibSetValue ]`

InhibitSetValue

Datentyp: `bool, num or dnum`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.305 TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt
RobotWare - OS
Fortsetzung

Der Name einer persistenten Variablen des Datentyps `bool`, `num` oder `dnum` oder jedes Alias dieser drei Basisdatentypen.

Das optionale Argument kann nur gemeinsam mit dem optionalen Argument `Inhib` verwendet werden.

Wenn dieses optionale Argument verwendet wird, und der Wert des Flags der persistenten Variablen, die im optionalen Argument `Inhib` verwendet wird, an der Positionszeit für die Signaleinstellung TRUE ist, wird der Wert der persistenten Variable gelesen, die im optionalen Argument `InhibSetValue` verwendet wird. Dieser Wert wird zur Einstellung von `DOp`, `GOp` oder des Signals `AOp` verwendet.

Bei Verwendung einer persistenten booleschen Variable wird der Wert TRUE in den Wert 1 und FALSE in den Wert 0 übertragen.

`[ \Mode ]`

Datentyp: `triggmode`

Wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

Programmabarbeitung

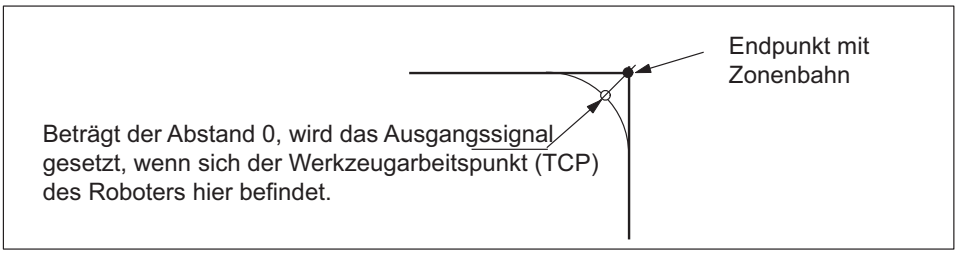
Beim Ausführen der Instruktion `TriggIO` wird die Triggerbedingung in einer angegebenen Variablen im Argument `TriggData` gespeichert.

Wenn danach eine der Instruktionen `TriggL`, `TriggC` oder `TriggJ` abgearbeitet wird, gilt hinsichtlich der Definitionen in `TriggIO` Folgendes:

Die folgende Tabelle beschreibt den im Argument `Distance` angegebenen Abstand:

Lineare Bewegung	Der geradlinige Abstand
Kreisbewegung	Die Länge des Kreisbogens
Nicht lineare Bewegung	Die ungefähre Bogenlänge entlang der Bahn (für angemessene Genauigkeit sollte der Abstand nicht die Hälfte der Bogenlänge überschreiten.)

Die Abbildung zeigt die positionsabhängige E/A-Prüfung auf einer Zonenbahn.



xx0500002248

Die positionsabhängige E/A-Prüfung wird generiert, wenn der Startpunkt (Endpunkt) passiert wird und der angegebene Abstand vom Endpunkt (Startpunkt) nicht innerhalb der Bewegungslänge der aktuellen Instruktion liegt (`Trigg...`).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte SetValue-Argument für das angegebene analoge Ausgangssignal AOp liegt außerhalb des zulässigen Bereichs.
ERR_GO_LIM	Das programmierte SetValue- oder SetDvalue-Argument für das angegebene digitale Gruppenausgangssignal GOp liegt außerhalb des zulässigen Bereichs.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion TriggIO sind unten aufgeführt.

Beispiel 1:

```
VAR triggdata glueflow;

TriggIO glueflow, 1 \Start \AOp:=glue, 5.3;

MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, glueflow, z50, tool1;
```

Das analoge Ausgangssignal glue wird auf den Wert5.3 gesetzt, wenn der Werkzeugarbeitspunkt (TCP) einen Punkt passiert, der sich 1 mm hinter dem Startpunkt p1 befindet.

Beispiel 2:

```
...
TriggL p3, v500, glueflow, z50, tool1;
```

Das analoge Ausgangssignal glue wird erneut auf den Wert 5.3 gesetzt, wenn der Werkzeugarbeitspunkt (TCP) einen Punkt passiert, der sich 1 mm hinter dem Startpunkt p2 befindet.

Einschränkungen

Abstandsabhängige E/A-Ereignisse (ohne das Argument \Time) sind für Fly-By-Punkte (Zonenbahn) gedacht. E/A-Ereignisse mit Abstand = 0 verzögern bei Verwendung von Stopppunkten das Auslösen, bis der Roboter den Punkt mit einer Genauigkeit von +/-24 ms erreicht hat.

Zeitabhängige E/A-Ereignisse (mit dem Argument \Time) sind für Stopppunkte vorgesehen. Zeitabhängige E/A-Ereignisse, die Fly-By-Punkte verwenden, führen zu schlechterer Genauigkeit als unten angegeben. Zeitabhängige E/A-Ereignisse können nur ab dem Endpunkt der Bewegung angegeben werden. Diese Zeit darf die aktuelle Bremszeit des Roboters nicht überschreiten, d. h. maximal ca. 0,5 s s (typische Werte bei der Geschwindigkeit 500 mm/s für IRB 2400 150 ms und für IRB 6400 250 ms). Wenn die angegebene Zeit die aktuelle Bremszeit überschreitet,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.305 TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt
RobotWare - OS
Fortsetzung

wird das E/A-Ereignis dennoch generiert, jedoch erst nach Beginn des Bremsvorgangs (also später als angegeben). Die gesamte Bewegungszeit für die aktuelle Bewegung kann während kurzer und schneller Bewegungen verwendet werden.

Die typische Absolutgenauigkeit für das Setzen digitaler Eingänge beträgt +/- 5 ms. Die typische Wiederholgenauigkeit für das Setzen digitaler Ausgänge beträgt +/-2 ms.

`TriggIO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggIO
  [TriggData ':='] <variable (VAR) of triggdata>','
  [Distance' :='] <expression (IN) of num>
  ['\' Start] | ['\' Time]
  ['\' DOp ':=' <variable (VAR) of signaldo>]
  |[ '\' GOp ':=' <variable (VAR) of signalgo>]
  |[ '\' AOp ':=' <variable (VAR) of signalao>]
  |[ '\' ProcID ':=' <expression (IN) of num>]','
  [SetValue ':='] <expression (IN) of num>
  |[ SetDvalue ':='] <expression (IN) of dnum>
  ['\' DODelay ':=' <expression (IN) of num>]
  ['\' Inhib ':=' <persistent (PERS) of bool>]
  ['\' InhibSetValue ':=' <persistent (PERS) of anytype>]
  ['\' Mode ':=' <expression (IN) of triggmode>]';'
```

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von positions-/zeitabhängigem E/A-Ereignis	*TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923*
Definition von positionsbezogenen Interrupts	*TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Speichern von Triggerdaten	*triggdata – Positioniereereignisse, trigg auf Seite 1833*
Definieren von verschiedenen Trigger-Aktionsmodi	*triggmode - Triggeraktionsmodus auf Seite 1840*
Definieren einer positionsabhängigen E/A-Prüfung	*TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.305 TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt

RobotWare - OS

Fortsetzung

Information über	Siehe
Setzen von E/A	*SetDO - Ändern des Werts eines digitalen Ausgangssignals auf Seite 710* *SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen auf Seite 713* *SetAO - Ändern des Werts eines analogen Ausgangssignals auf Seite 700*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.306 TriggJ - Achsenweise Roboterbewegungen mit Ereignissen

Verwendung

`TriggJ` (*TriggJoint*) wird verwendet, um Ausgangssignale und/oder Interrupt-Routinen an ungefähr festgelegten Positionen zu setzen, während der Roboter sich rasch von einem Punkt zu einem anderen bewegt, wobei diese Bewegung nicht gradlinig erfolgen muss.

Eines oder mehrere (maximal 25) Ereignisse können mithilfe der Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert werden. Anschließend kann mit der Instruktion `TriggJ` auf diese Definitionen verwiesen werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggJ`:

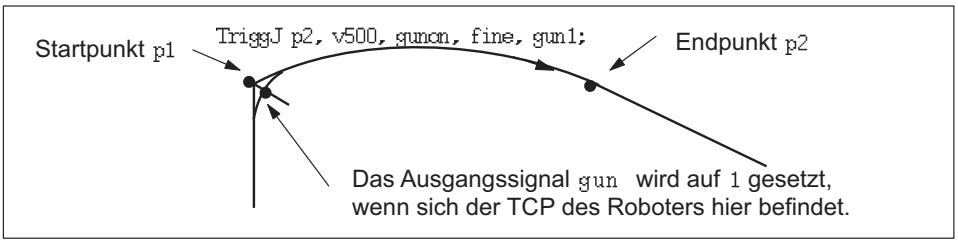
Siehe auch *Weitere Beispiele auf Seite 947*.

Beispiel 1:

```
VAR triggdata gunon;
...
TriggIO gunon, 0 \Start \DOp:=gun, 1;
MoveL p1, v500, z50, gun1;
TriggJ p2, v500, gunon, fine, gun1;
```

Das digitale Ausgangssignal `gun` wird gesetzt, wenn der TCP die Mitte der Zonenbahn des Punkts `p1` passiert.

Die Abbildung zeigt ein Beispiel für ein positionsabhängiges E/A-Ereignis.



xx0500002272

Argumente

```
TriggJ [\Conc] ToPoint [\ID] Speed [\T] Trigg_1 | TriggArray [ \T2
     ] [ \T3 ] [\T4] [\T5] [\T6] [\T7] [\T8] Zone [\Inpos] Tool
     [\WObj] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp:`switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Das Argument kann bei der Verwendung von Fly-By-Punkten verwendet werden,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

um unerwünschte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden.

Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss. Es kann zudem zur Abstimmung der Abarbeitung mit der Roboterbahn verwendet werden, um die Warnung „50024 Corner path failure“ oder den Fehler „50082 Declaration limit“ zu vermeiden.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Wenn dieses Argument ausgelassen wird, wird die nachfolgende Instruktion erst abgearbeitet, nachdem der Roboter den angegebenen Stopppunkt erreicht hat oder wenn er sich 100 ms vor der angegebenen Zone befindet.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToPoint`

Datentyp:`robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der Werkzeugumorientierung und der externen Achsen.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.306 TriggJ - Achsenweise Roboterbewegungen mit Ereignissen
RobotWare - OS
Fortsetzung

beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`Trigg_1`

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`TriggArray`

Trigg Data Array Parameter

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm definiert wurden mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO`.

Die Beschränkung beträgt 25 Elemente im Array, und 1 bis 25 Triggerbedingungen müssen definiert werden.

Es ist nicht möglich, die optionalen Argumente `T2`, `T3`, `T4`, `T5`, `T6`, `T7` oder `T8` zur gleichen Zeit zu verwenden, wenn das `TriggArray`-Argument verwendet wird.

`[ \T2 ]`

Trigg 2

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T3 ]`

Trigg 3

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T4 ]`

Trigg 4

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T5 ]`

Trigg 5

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \T6 ]`

Trigg 6

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T7 ]`

Trigg 7

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`[ \T8 ]`

Trigg 8

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggCheckIO`, `TriggSpeed` oder `TriggRampAO` definiert wurde.

`Zone`

Datentyp:`zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Inpos ]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine Achsenbewegung in Relation zum Werkobjekt möglich ist.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente `loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Siehe die Instruktion `MoveJ` für weitere Informationen über Achsenbewegungen.

Wenn die Triggerbedingungen erfüllt sind, während sich der Roboter dem Endpunkt nähert, werden die definierten Triggeraktivitäten ausgeführt. Die Triggerbedingungen sind entweder einen bestimmten Abstand vor dem Endpunkt der Instruktion oder an einem eingestellten Abstand nach dem Startpunkt der Instruktion oder zu einem bestimmten Zeitpunkt (auf eine kurze Zeit begrenzt) vor dem Endpunkt der Instruktion erfüllt.

Bei der schrittweisen Vorwärtsabarbeitung werden die E/A-Aktivitäten ausgeführt, aber die Interrupt-Routinen werden nicht abgearbeitet. Bei der schrittweisen Rückwärtsabarbeitung werden überhaupt keine Triggeraktivitäten ausgeführt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte ScaleValue-Argument für das angegebene analoge Ausgangssignal AOp liegt in einigen der verbundenen TriggSpeed-Instruktionen zusammen mit dem programmierten Speed-Argument in dieser Instruktion außerhalb der Grenzen für das analoge Signal.
ERR_DIPLAG_LIM	Das programmierte DipLag-Argument in einer verbundenen TriggSpeed-Instruktion ist im Verhältnis zur verwendeten Event Preset Time in den Systemparametern zu groß.
ERR_NORUNUNIT	Beim Einstieg in die Instruktion kein Kontakt mit dem E/A-Gerät vorhanden ist und die verwendeten triggdata erfordern, dass das E/A-Gerät in Betrieb ist (d. h. dass in den triggdata ein Signal verwendet wird).
ERR_CONC_MAX	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument \Conc verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion TriggJ sind unten aufgeführt.

Beispiel 1:

```
VAR intnum intno1;
VAR triggdata trigg1;
...
PROC main()
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 0.1 \Time, intno1;
  ...
  TriggJ p1, v500, trigg1, fine, gun1;
  TriggJ p2, v500, trigg1, fine, gun1;
  ...
  IDelete intno1;
```

Die Interrupt-Routine trap1 wird ausgeführt, wenn sich der Werkzeugarbeitspunkt an einer Position 0,1 s vor dem Stopppunkt p1 bzw. p2 befindet.

Beispiel 2:

```
VAR num Distance:=0;
VAR triggdata trigg_array{25};
VAR signaldo myaliassignaldo;
VAR string signalname;
...
PROC main()
  ...
  FOR i FROM 1 TO 25 DO
    signalname:="do";
    signalname:=signalname+ValToStr(i);
    AliasIO signalname, myaliassignaldo;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.306 TriggJ - Achsenweise Roboterbewegungen mit Ereignissen

RobotWare - OS

Fortsetzung

```
    TriggEquip trigg_array{i}, Distance \Start, 0
        \DOp:=myaliassignaldo, SetValue:=1;
    Distance:=Distance+10;
  ENDFOR
  TriggJ p1, v500, trigg_array, z30, tool2;
  MoveJ p2, v500, z30, tool2;
  ...
```

Die digitalen Ausgangssignale `do1` bis `do25` werden während der Bewegung zu `p1` festgelegt. Der Abstand zwischen den Signaleinstellungen beträgt 10 mm.

Einschränkungen

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion `TriggJ` kürzer als normal ist (z. B. am Start von `TriggJ` mit der Roboterposition am Endpunkt), sind eventuell einige oder alle Triggerbedingungen sofort und an derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine unvollständige Bewegung nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

`TriggJ` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggJ
  [ '\' Conc ',']
  [ ToPoint' :=' ] < expression (IN) of robtarget >
  [ '\' ID ':=' < expression (IN) of identno >]','
  [ Speed ':=' ] < expression (IN) of speeddata >
  [ '\' T ':=' < expression (IN) of num > ]','
  [Trigg_1 ':=' ] < variable (VAR) of triggdata > |
  [TriggArray ':=' ] < array variable {*} (VAR) of triggdata >
  [ '\' T2 ':=' < variable (VAR) of triggdata > ]
  [ '\' T3 ':=' < variable (VAR) of triggdata > ]
  [ '\' T4 ':=' < variable (VAR) of triggdata > ]
  [ '\' T5 ':=' < variable (VAR) of triggdata > ]
  [ '\' T6 ':=' < variable (VAR) of triggdata > ]
  [ '\' T7 ':=' < variable (VAR) of triggdata > ]
  [ '\' T8 ':=' < variable (VAR) of triggdata > ] ','
  [Zone ':=' ] < expression (IN) of zonedata >
  [ '\' Inpos ':=' < expression (IN) of stoppointdata > ]','
  [ Tool ':=' ] < persistent (PERS) of tooldata >
  [ '\' WObj' :='< persistent (PERS) of wobjdata > ]
  [ '\' TLoad' :=' < persistent (PERS) of loaddata > ] ';'
```

Weitere Informationen

Information über	Siehe
Lineare Bewegung mit Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Kreisförmige Bewegung mit Triggern	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913*
Handhabung `triggdata`	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen auf Seite 921* *TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable auf Seite 919* *TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist auf Seite 1563*
Achsenbewegung des Roboters	*MoveJ - Achsenbewegung des Roboters auf Seite 422*
Achsenbewegung	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.307 TriggL - Lineare Roboterbewegungen mit Ereignissen

Verwendung

`TriggL` (*Trigg Linear*) wird verwendet, um Ausgangssignale und/oder Interrupt-Routinen an festen Positionen zu setzen, während der Roboter lineare Bewegungen durchführt.

Eines oder mehrere (maximal 25) Ereignisse können mithilfe der Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert werden. Anschließend kann mit der Instruktion `TriggL` auf diese Definitionen verwiesen werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggL`:

Siehe auch *Weitere Beispiele auf Seite 955*.

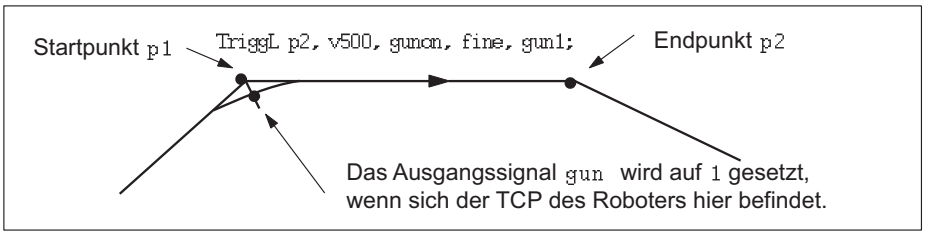
Beispiel 1:

```
VAR triggdata gunon;

TriggIO gunon, 0 \Start \DOp:=gun, 1;
MoveJ p1, v500, z50, gun1;
TriggL p2, v500, gunon, fine, gun1;
```

Das digitale Ausgangssignal `gun` wird gesetzt, wenn der TCP die Mitte der Zonenbahn des Punkts `p1` passiert.

Die Abbildung zeigt ein Beispiel für ein positionsabhängiges E/A-Ereignis.



xx0500002291

Argumente

```
TriggL [\Conc] ToPoint [\ID] Speed [\T] Trigg_1 | TriggArray [\T2]
     [\T3] [\T4] [\T5] [\T6] [\T7] [\T8] Zone [\Inpos] Tool [\WObj]
     [\Corr] [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp:`switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Das Argument kann bei der Verwendung von Fly-By-Punkten verwendet werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden.

Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss. Es kann zudem zur Abstimmung der Abarbeitung mit der Roboterbahn verwendet werden, um die Warnung „50024 Corner path failure“ oder den Fehler „50082 Declaration limit“ zu vermeiden.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToPoint`

Datentyp:`robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der externen Achsen und der Werkzeugumorientierung.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.307 TriggL - Lineare Roboterbewegungen mit Ereignissen

RobotWare - OS

Fortsetzung

`Trigg_1`

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`TriggArray`

Trigg Data Array Parameter

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm definiert wurden mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO`.

Die Beschränkung beträgt 25 Elemente im Array, und 1 bis 25 Triggerbedingungen müssen definiert werden.

Es ist nicht möglich, die optionalen Argumente `T2`, `T3`, `T4`, `T5`, `T6`, `T7` oder `T8` zur gleichen Zeit zu verwenden, wenn das `TriggArray`-Argument verwendet wird.

`[ \T2 ]`

Trigg 2

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`[ \T3 ]`

Trigg 3

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`[ \T4 ]`

Trigg 4

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`[ \T5 ]`

Trigg 5

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \T6 ]`

Trigg 6

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`[ \T7 ]`

Trigg 7

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`[ \T8 ]`

Trigg 8

Datentyp: `triggdata`

Variable, die auf Triggerbedingungen und Triggeraktivitäten verweist und zuvor im Programm mit den Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` definiert wurde.

`Zone`

Datentyp:`zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Inpos ]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \Corr ]`

Correction

Datentyp: `switch`

Korrekturdaten, die durch die Instruktion `CorrWrite` in einen Korrektureintrag geschrieben wurden, werden der Bahn und der Zielposition hinzugefügt, falls dieses Argument vorhanden ist.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Programmabarbeitung

Siehe die Instruktion `MoveL` für weitere Informationen über lineare Bewegung.

Wenn die Triggerbedingungen erfüllt sind, während sich der Roboter dem Endpunkt nähert, werden die definierten Triggeraktivitäten ausgeführt. Die

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Triggerbedingungen sind entweder einen bestimmten Abstand vor dem Endpunkt der Instruktion oder an einem eingestellten Abstand nach dem Startpunkt der Instruktion oder zu einem bestimmten Zeitpunkt (auf eine kurze Zeit begrenzt) vor dem Endpunkt der Instruktion erfüllt.

Bei der schrittweisen Vorwärtsabarbeitung werden die E/A-Aktivitäten ausgeführt, aber die Interrupt-Routinen laufen nicht ab. Bei der schrittweisen Rückwärtsabarbeitung werden überhaupt keine Triggeraktivitäten ausgeführt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte ScaleValue-Argument für das angegebene analoge Ausgangssignal AOp liegt in einigen der verbundenen TriggSpeed-Instruktionen zusammen mit dem programmierten Speed-Argument in dieser Instruktion außerhalb der Grenzen für das analoge Signal.
ERR_DIPLAG_LIM	Das programmierte DipLag-Argument in einer verbundenen TriggSpeed-Instruktion ist im Verhältnis zur verwendeten Event Preset Time in den Systemparametern zu groß.
ERR_NORUNUNIT	Beim Einstieg in die Instruktion kein Kontakt mit dem E/A-Gerät vorhanden ist und die verwendeten triggdata erfordern, dass das E/A-Gerät in Betrieb ist (d. h. dass in den triggdata ein Signal verwendet wird).
ERR_CONC_MAX	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument \Conc verwenden, wurde überschritten.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion TriggL sind unten aufgeführt.

Beispiel 1:

```
VAR intnum intno1;
VAR triggdata trigg1;
...
PROC main()
  CONNECT intno1 WITH trap1;
  TriggInt trigg1, 0.1 \Time, intno1;
  ...
  TriggL p1, v500, trigg1, fine, gun1;
  TriggL p2, v500, trigg1, fine, gun1;
  ...
  IDelete intno1;
```

Die Interrupt-Routine trap1 wird ausgeführt, wenn sich der Werkzeugarbeitspunkt an einer Position 0.1 s vor dem Punkt p1 oder p2 befindet.

Beispiel 2:

```
VAR num Distance:=0;
VAR triggdata trigg_array{25};
VAR signaldo myaliassignaldo;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.307 TriggL - Lineare Roboterbewegungen mit Ereignissen

RobotWare - OS

Fortsetzung

```
VAR string signalname;
...
PROC main()
  ...
  FOR i FROM 1 TO 25 DO
    signalname:="do";
    signalname:=signalname+ValToStr(i);
    AliasIO signalname, myaliassignaldo;
    TriggEquip trigg_array{i}, Distance \Start, 0
         \DOp:=myaliassignaldo, SetValue:=1;
    Distance:=Distance+10;
  ENDFOR
  TriggL p1, v500, trigg_array, z30, tool2;
  MoveL p2, v500, z30, tool2;
  ...
```

Die digitalen Ausgangssignale `do1` bis `do25` werden während der Bewegung zu `p1` festgelegt. Der Abstand zwischen den Signaleinstellungen beträgt 10 mm.

Einschränkungen

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion `TriggL` kürzer als normal ist (z. B. am Start von `TriggL` mit der Roboterposition am Endpunkt), sind eventuell einige oder alle Triggerbedingungen sofort und an derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine unvollständige Bewegung nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

`TriggL` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggL
  ['\' Conc ',']
  [ToPoint' :='] <expression (IN) of robtarget>
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' T ':=' <expression (IN) of num>]','
  [Trigg_1 ':='] <variable (VAR) of triggdata>|
  [TriggArray ':='] <array variable {*} (VAR) of triggdata>
  ['\' T2 ':=' <variable (VAR) of triggdata>]
  ['\' T3 ':=' <variable (VAR) of triggdata>]
  ['\' T4 ':=' <variable (VAR) of triggdata>]
  ['\' T5 ':=' <variable (VAR) of triggdata>]
  ['\' T6 ':=' <variable (VAR) of triggdata>]
  ['\' T7 ':=' <variable (VAR) of triggdata>]
  ['\' T8 ':=' <variable (VAR) of triggdata>]','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':='] <persistent (PERS) of tooldata>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
['\' WObj ':=' <persistent (PERS) of wobjdata>]
['\' Corr]
['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Kreisförmige Bewegung mit Triggern	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Achsenbewegung mit Triggern	*TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung auf Seite 983*
Handhabung `triggdata`	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen auf Seite 921* *TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable auf Seite 919* *TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist auf Seite 1563*
Schreiben an einen Korrektureintrag	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lineare Bewegung	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Definition von Zonendaten	*zonedata - Zonendaten auf Seite 1860*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.307 TriggL - Lineare Roboterbewegungen mit Ereignissen
RobotWare - OS
Fortsetzung

Information über	Siehe
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.308 TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen

Verwendung

`TriggJIOs` (*Trigg Joint I/O*) wird verwendet, um Ausgangssignale an festen Positionen zu setzen, während der Roboter Gelenkbewegungen durchführt.

Die `TriggJIOs`-Instruktion liefert eine hohe Genauigkeit, wenn Bewegungen mit Zonen verwendet werden (im Vergleich zu `TriggEquip`/`TriggL`).

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggJIOs`:

Siehe auch *Weitere Beispiele auf Seite 972*.

Beispiel 1:

```
VAR triggios gunon{1};

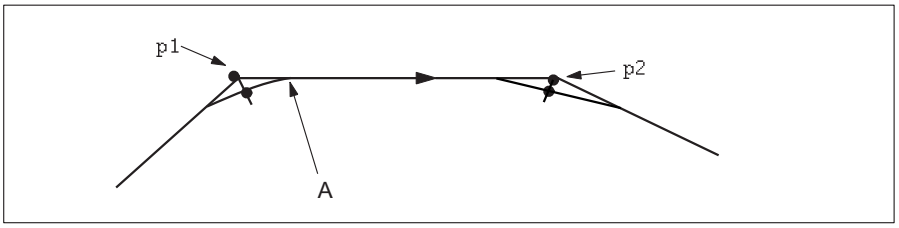
gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=1;

MoveJ p1, v500, z50, gun1;
TriggJIOs p2, v500, \TriggData1:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das Signal `gun` wird gesetzt, wenn sich der TCP 3 mm hinter dem Punkt `p1` befindet.

RAPID-Code und -Abbildung zeigen ein Beispiel für ein positionsabhängiges E/A-Ereignis.

```
TriggJIOs p2, v500, \TriggData1:=gunon, z50, gun1;
```



xx1500000304

A Das Ausgangssignal `gun` wird auf `1` gesetzt, wenn der TCP des Roboters da ist.

Argumente

```
TriggJIOs [\Conc] ToPoint [\ID] Speed [\T] [\TriggData1]
     [\TriggData2] [\TriggData3] Zone [\Inpos] Tool [\WObj]
     [\TLoad]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.308 TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen
RobotWare - OS
Fortsetzung

`[ \Conc ]`

Concurrent

Datentyp:`switch`

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Das Argument kann bei der Verwendung von Fly-By-Punkten verwendet werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden.

Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss. Es kann zudem zur Abstimmung der Abarbeitung mit der Roboterbahn verwendet werden, um die Warnung „50024 Corner path failure“ oder den Fehler „50082 Declaration limit“ zu vermeiden.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToPoint`

Datentyp:`robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der externen Achsen und der Werkzeugumorientierung.

`[ \T ]`

Time

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`[\TriggData1]`

Datentyp: `array of triggios`

Variable (Datenfeld), die auf Triggerbedingungen und Triggeraktivitäten verweist. Bei Verwendung dieses Arguments können analoge Ausgangssignale, digitale Ausgangssignale und digitale Gruppenausgangssignale gesetzt werden. Wenn ein digitales Gruppenausgangssignal gesetzt wird, ist die Anzahl der Signale in der Gruppe auf 23 Signale begrenzt.

`[\TriggData2]`

Datentyp: `array of triggstrgo`

Variable (Datenfeld), die auf Triggerbedingungen und Triggeraktivitäten verweist. Wenn dieses Argument verwendet wird, können digitale Gruppenausgangssignale gesetzt werden, die aus 32 Signalen in der Gruppe bestehen und einen maximalen Sollwert von 4294967295 aufweisen. Es können nur digitale Gruppenausgangssignale verwendet werden.

`[\TriggData3]`

Datentyp: `array of triggiosdnum`

Variable (Datenfeld), die auf Triggerbedingungen und Triggeraktivitäten verweist. Wenn dieses Argument verwendet wird, können analoge Ausgangssignale, digitale Ausgangssignale und digitale Gruppenausgangssignale gesetzt werden, die aus 32 Signalen in der Gruppe bestehen und einen maximalen Sollwert von 4294967295 aufweisen.

`Zone`

Datentyp:`zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Inpos ]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.308 TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen
RobotWare - OS
Fortsetzung

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Siehe Anweisung `MoveJ` für weitere Informationen über Gelenkbewegungen, *MoveJ - Achsenbewegung des Roboters auf Seite 422*.

Mit der Instruktion `TriggJIOs` können 1-50 unterschiedliche Triggeraktivitäten für E/A-Signale auf einer Bahn von A nach B eingerichtet werden. Es können digitale Ausgangssignale, digitale Gruppenausgangssignale und analoge Ausgangssignale verwendet werden Die Triggerbedingungen werden entweder in einer bestimmten Entfernung vor dem Endpunkt der Instruktion oder in einer bestimmten Entfernung nach dem Anfangspunkt der Instruktion erfüllt.

Für die Instruktion muss eines der Argumente `TriggData1`, `TriggData2` oder `TriggData3` oder alle drei verwendet werden. Die Verwendung anderer Triggerargumente ist optional. Um die Verwendung eines Triggers zu verhindern, kann die Komponente `used` in der Datenfeldkomponente der Datentypen `triggios`/`triggstrgo`/`triggiosdnum` auf FALSE gesetzt werden. Wenn keine Datenfeldkomponente verwendet wird, entspricht das Verhalten der `TriggJIOs`-Instruktion einer `MoveJ`, und es werden keine E/A-Aktivitäten ausgeführt.

Bei der Vorwärtsabarbeitung des Programms werden die E/A-Aktivitäten ausgeführt. Bei der schrittweisen Rückwärtsabarbeitung werden keine E/A-Aktivitäten ausgeführt.

Wenn die Komponente `EquipLag` im `TriggData1`-, `TriggData2` oder `TriggData3`-Argument auf einen negativen Zeitwert (Verzögerung) gesetzt wird, kann das E/A-Signal hinter dem Zielpunkt (`ToPoint`) gesetzt werden.

Bei Verwendung des Arguments `TriggData2` or `TriggData3` können Werte bis 4294967295 verwendet werden. Dies ist der maximal zulässige Wert für eine Gruppe digitaler Signale (ein Gruppensignal im System darf maximal 32 Signale enthalten).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_GO_LIM`	Das programmierte `setvalue`-Argument für das angegebene digitale Gruppenausgangssignal `signalname` liegt außerhalb des zulässigen Bereichs. (Deklariert in `TriggData1`, `TriggData2` oder `TriggData3`.)
`ERR_AO_LIM`	Das programmierte `setvalue`-Argument für das angegebene analoge Ausgangssignal `signalname` liegt außerhalb des zulässigen Bereichs. (Deklariert in `TriggData1` oder `TriggData3`.)

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.308 TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen
RobotWare - OS
Fortsetzung

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TriggJIOs` sind unten aufgeführt.

Beispiel 1:

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggJIOs p2, v500, \TriggData1:=mytriggios, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das digitale Gruppenausgangssignal `go1` wird in einer Entfernung von 3 mm von `p1` auf den Wert 55 gesetzt. Das analoge Ausgangssignal wird in einer Entfernung von 15 mm von `p1` auf den Wert 10 gesetzt. Das digitale Ausgangssignal do1 wird in einer Entfernung von 3 mm von ToPoint `p2` gesetzt.

Beispiel 2:

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
VAR triggstrgo mytriggstrgo{3}:= [[TRUE, 3, TRUE, 0, "go2", "1",
     0], [TRUE, 15, TRUE, 0, "go2", "800000", 0], [TRUE, 4, FALSE,
     0, "go2", "4294967295", 0]];
VAR triggiosdnum mytriggiosdnum{3}:= [[TRUE, 10, TRUE, 0, "go3",
     4294967295, 0], [TRUE, 10, TRUE, 0, "ao2", 5, 0], [TRUE, 10,
     TRUE, 0, "do2", 1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggJIOs p2, v500, \TriggData1:=mytriggios \TriggData2:=
     mytriggstrgo \TriggData3:=mytriggiosdnum, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das digitale Gruppenausgangssignal `go1` wird in einer Entfernung von 3 mm von `p1` auf den Wert 55 gesetzt. Das analoge Ausgangssignal `ao1` wird in einer Entfernung von 15 mm von `p1` auf den Wert 10 gesetzt. Das digitale Ausgangssignal `do1` wird in einer Entfernung von 3 mm von `ToPoint p2` gesetzt. Diese Positionsereignisse werden von der Variablen `mytriggios` festgelegt. Die Variable `mytriggstrgo` legt fest, dass Positionsereignisse in einer Entfernung von 3 und 15 mm von `p1` eintreten. Zunächst wird das Signal `go2` auf 1 und dann auf 800000 gesetzt. Das Signal wird in einer Entfernung von 4 mm vom `ToPoint p2` auf 4294967295 gesetzt. Dies ist der Höchstwert für ein digitales Ausgangssignal mit 32 Bit. Die Variable `mytriggiosdnum` legt fest, dass drei Positionsereignisse in einer Entfernung von 10 mm von `p1` eintreten. Zunächst wird das Signal `go3` auf 4294967295 gesetzt, dann wird `ao2` auf 5 gesetzt, und schließlich wird `do2` auf 1 gesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion `TriggJIOs` kürzer als normal ist (z. B. am Start von `TriggJIOs` mit der Roboterposition am Endpunkt), sind eventuell einige oder alle Triggerbedingungen sofort und an derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine unvollständige Bewegung nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

Die Begrenzung der Triggeranzahl in der Instruktion `TriggJIOs` ist 50 für jede programmierte Instruktion. Wenn Trigger bei einer geringeren Entfernung auftreten, könnte es sein, dass das System diese nicht behandelt. Das hängt von der Art ab, in der die Bewegung durchgeführt wurde, von der benutzten TCP Geschwindigkeit, und davon, wie nah die Trigger programmiert sind. Diese Einschränkungen existieren, aber es ist schwer vorauszusagen, wann solche Probleme auftreten.

`TriggJIOs` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Begrenzungen bezüglich der Genauigkeit

Abstandsabhängige E/A-Ereignisse sind für Fly-By-Punkte (Zonenbahn) gedacht. Die Verwendung von Stopppunkten führt zu einer geringeren Genauigkeit als unten angegeben.

Hinsichtlich der Genauigkeit für E/A-Ereignisse mit Abstand und Fly-By-Punkten (Bahnverlauf) gilt Folgendes, wenn ein digitaler Ausgang im angegebenen Abstand vom Start- oder Endpunkt in der Instruktion `TriggLIOs` oder `TriggCIOs` gesetzt wird:

- Die nachstehend angegebene Geschwindigkeit gilt bei Verwendung eines positiven `equiplag`[I] der weniger als 40 ms ist, was der Verzögerung im Roboterservo ohne Änderung des Systemparameters *Event Preset Time* entspricht. Die Verzögerung kann je nach Robotertyp variieren.
- Die nachfolgend angegebene Genauigkeit gilt bei Verwendung eines positiven `equiplag`, der weniger als den konfigurierten *Event Preset Time* in den Systemparametern entspricht.
- Die unten angegebene Genauigkeit gilt nicht für einen positiven `equiplag`, der größer als der in den Systemparametern konfigurierte *Event Preset Time* ist. In diesem Fall wird eine Näherungsmethode verwendet, bei der die dynamischen Grenzen des Roboters nicht berücksichtigt werden. Dann muss `SingArea \Wrist` verwendet werden, um eine akzeptable Genauigkeit zu erzielen.
- Die nachstehend angegebene Geschwindigkeit gilt bei Verwendung eines negativen `equiplag`.

I `equiplag` ist ein Datenobjekt des Datentyps `triggios`

Die typischen absoluten Genauigkeitswerte zur Einstellung digitaler Ausgänge ist: ±5 ms.

Die typischen wiederholten Genauigkeitswerte zur Einstellung digitaler Ausgänge ist: ±2 ms.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.308 TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen

RobotWare - OS

Fortsetzung

Syntax

```
TriggJIOs
  ['\' Conc ',']
  [ToPoint ':='] <expression (IN) of robtarget>
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' T ':=' <expression (IN) of num>]','
  ['\' TriggData1 ':='] <array {*} (VAR) of triggios>
  ['\' TriggData2 ':='] <array {*} (VAR) of triggstrgo>
  ['\' TriggData3 ':='] <array {*} (VAR) of triggiosdnum>','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':=' ] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Lineare Roboterbewegungen mit E/A-Ereignissen	*TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen auf Seite 967*
Speicherung von Triggerbedingungen und Triggeraktivitäten	*triggios – Positioniereereignisse, trigg auf Seite 1835*
Speicherung von Triggerbedingungen und Triggeraktivitäten für digitale Signalgruppen, die aus 32 Signalen bestehen	*triggstrgo - Positioning events, trigg auf Seite 1843*
Speicherung von Triggerbedingungen und Triggeraktivitäten	*triggiosdnum – Positioniereereignisse, trigg auf Seite 1838*
Zuweisung von Ereignisobjekten	*Technisches Referenzhandbuch - Systemparameter*
Lineare Bewegung	*Technisches Referenzhandbuch - RAPID Overview*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.309 TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen

Verwendung

`TriggLIOs` (Trigger Lineare E/A) wird verwendet, um Ausgangssignale an festen Positionen zu setzen, während der Roboter lineare Bewegungen durchführt.

Die `TriggLIOs`-Instruktion liefert eine hohe Genauigkeit, wenn Bewegungen mit Zonen verwendet werden (im Vergleich zu `TriggEquip`/`TriggL`).

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggLIOs`:

Siehe auch *Weitere Beispiele auf Seite 972*.

Beispiel 1:

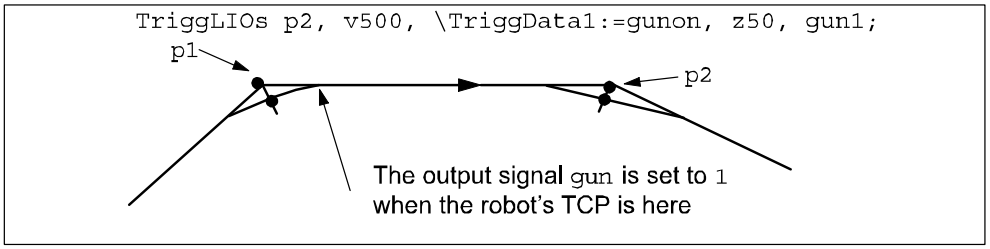
```
VAR triggios gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=1;

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das Signal `gun` wird gesetzt, wenn sich der TCP 3 mm hinter dem Punkt `p1` befindet.

Die Abbildung zeigt ein Beispiel für ein positionsabhängiges E/A-Ereignis.



en0800000157

Argumente

```
TriggLIOs [\Conc] ToPoint [\ID] Speed [\T] [\TriggData1]
     [\TriggData2] [\TriggData3] Zone [\Inpos] Tool [\WObj] [\Corr]
     [\TLoad]
```

`[ \Conc ]`

Concurrent

Datentyp:`switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.309 TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen
RobotWare - OS
Fortsetzung

Nachfolgende Instruktionen werden abgearbeitet, während sich der Roboter bewegt. Das Argument kann bei der Verwendung von Fly-By-Punkten verwendet werden, um nicht gewollte Stopps zu vermeiden, die durch eine überlastete CPU ausgelöst werden. Das ist sehr vorteilhaft, wenn die programmierten Punkte sehr nahe aneinander liegen und hohe Geschwindigkeiten gefahren werden.

Das Argument ist außerdem sehr nützlich, wenn z. B. mit externen Geräten kommuniziert wird und keine Synchronisierung zwischen den externen Geräten und den Roboterbewegungen stattfinden muss. Es kann zudem zur Abstimmung der Abarbeitung mit der Roboterbahn verwendet werden, um die Warnung „50024 Corner path failure“ oder den Fehler „50082 Declaration limit“ zu vermeiden.

Mit dem Argument `\Conc` ist die Anzahl der aufeinander folgenden Bewegungsinstruktionen auf 5 beschränkt. In einem Programmabschnitt, der `StorePath-RestoPath` enthält, sind Bewegungsinstruktionen mit dem Argument `\Conc` nicht zulässig.

Wenn dieses Argument ausgelassen wird und der `ToPoint` kein Stopppunkt ist, wird die nachfolgende Instruktion abgearbeitet, bevor der Roboter die programmierte Zone erreicht hat.

Dieses Argument kann nicht in koordinierten synchronisierten Bewegungen in einem MultiMove-System verwendet werden.

`ToPoint`

Datentyp:`robtarget`

Der Zielpunkt des Roboters und der externen Achsen. Er wird als benannte Position definiert oder direkt in der Instruktion gespeichert (mit einem * gekennzeichnet).

`[ \ID ]`

Synchronization id

Datentyp: `identno`

Das Argument `[ \ID ]` ist bei MultiMove-Systemen obligatorisch, wenn die Bewegung synchronisiert oder koordiniert synchronisiert ist. Dieses Argument ist in einem anderen Fall nicht zulässig. Die angegebene ID-Nummer muss für alle kooperierenden Programmtasks identisch sein. Durch die ID-Nummer werden Bewegungen zur Laufzeit nicht verwechselt.

`Speed`

Datentyp: `speeddata`

Die Geschwindigkeitsdaten, die für Bewegungen gelten. Geschwindigkeitsdaten definieren die Geschwindigkeit des Werkzeugarbeitspunkts, der externen Achsen und der Werkzeugumorientierung.

`[ \T ]`

Time

Datentyp: `num`

Dieses Argument legt die Gesamtzeit in Sekunden fest, während der sich der Roboter bewegt. Es wird durch die entsprechenden Geschwindigkeitsdaten ersetzt. Die Geschwindigkeitsdaten werden unter der Annahme berechnet, dass die Geschwindigkeit während der Bewegung konstant ist. Wenn der Roboter diese

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Geschwindigkeit nicht während der gesamten Bewegung aufrechterhalten kann, beispielsweise wenn die Bewegung an einem Feinpunkt beginnt oder endet, dann wird die tatsächliche Bewegungszeit die programmierte Zeit überschreiten.

`[\TriggData1]`

Datentyp: `array of triggios`

Variable (Datenfeld), die auf Triggerbedingungen und Triggeraktivitäten verweist. Bei Verwendung dieses Arguments können analoge Ausgangssignale, digitale Ausgangssignale und digitale Gruppenausgangssignale gesetzt werden. Wenn ein digitales Gruppenausgangssignal gesetzt wird, ist die Anzahl der Signale in der Gruppe auf 23 Signale begrenzt.

`[\TriggData2]`

Datentyp: `array of triggstrgo`

Variable (Datenfeld), die auf Triggerbedingungen und Triggeraktivitäten verweist. Wenn dieses Argument verwendet wird, können digitale Gruppenausgangssignale gesetzt werden, die aus 32 Signalen in der Gruppe bestehen und einen maximalen Sollwert von 4294967295 aufweisen. Es können nur digitale Gruppenausgangssignale verwendet werden.

`[\TriggData3]`

Datentyp: `array of triggiosdnum`

Variable (Datenfeld), die auf Triggerbedingungen und Triggeraktivitäten verweist. Wenn dieses Argument verwendet wird, können analoge Ausgangssignale, digitale Ausgangssignale und digitale Gruppenausgangssignale gesetzt werden, die aus 32 Signalen in der Gruppe bestehen und einen maximalen Sollwert von 4294967295 aufweisen.

`Zone`

Datentyp:`zonedata`

Zonendaten für die Bewegung. Zonendaten beschreiben die Größe der erzeugten Zonenbahn.

`[ \Inpos ]`

In position

Datentyp: `stoppointdata`

Dieses Argument wird verwendet, um die Annäherungskriterien für die Position des Roboter-TCP am Stopppunkt anzugeben. Die Daten des Stopppunkts ersetzen die Zone, die im Parameter `Zone` angegeben ist.

`Tool`

Datentyp: `tooldata`

Das verwendete Werkzeug, wenn sich der Roboter bewegt. Der Werkzeugarbeitspunkt ist der Punkt, der an die angegebene Zielposition bewegt wird.

`[ \WObj ]`

Work Object

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.309 TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen
RobotWare - OS
Fortsetzung

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht.

Dieses Argument kann ausgelassen werden; in diesem Fall bezieht sich die Position auf das Welt-Koordinatensystem. Wenn allerdings ein stationärer TCP oder koordinierte externe Achsen verwendet werden, muss dieses Argument angegeben werden, damit eine lineare Bahn in Relation zum Werkobjekt möglich ist.

`[ \Corr ]`

Correction

Datentyp: `switch`

Korrekturdaten, die durch die Instruktion `CorrWrite` in einen Korrektureintrag geschrieben wurden, werden der Bahn und der Zielposition hinzugefügt, falls dieses Argument vorhanden ist.

Die RobotWare-Option *Path Offset* ist bei der Verwendung dieses Arguments erforderlich.

`[ \TLoad ]`

Total load

Datentyp: `loaddata`

Das Argument `\TLoad` beschreibt die in der Bewegung verwendete Gesamtlast. Die Gesamtlast ist die Summe der Werkzeuglast und der Nutzlast des Werkzeugs. Bei Verwendung des Arguments `\TLoad` werden die `loaddata` in den aktuellen `tooldata` nicht berücksichtigt.

Wenn das `\TLoad`-Argument auf `load0` gesetzt ist, wird das `\TLoad`-Argument nicht berücksichtigt, und stattdessen werden die `loaddata` in den aktuellen `tooldata` verwendet.

Damit das `\TLoad`-Argument verwendet werden kann, muss der Wert des Systemparameters `ModalPayLoadMode` unbedingt auf 0 gesetzt werden. Wird `ModalPayLoadMode` auf 0 eingestellt, kann die `GripLoad`-Instruktion nicht mehr genutzt werden.

Die Gesamtlast kann mit der Serviceroutine LoadIdentify ermittelt werden. Wurde der Systemparameter `ModalPayLoadMode` auf 0 gesetzt, kann der Bediener beim Ausführen der Serviceroutine die `loaddata` vom Werkzeug in eine bereits vorhandene oder eine neue persistente`loaddata`-Variable kopieren.

Es ist möglich, über ein digitales Eingangssignal, das mit dem Systemeingang `SimMode` (Simulationsmodus) verbunden ist, einen Testlauf des Programms ohne Nutzlast durchzuführen. Wenn das digitale Eingangssignal auf 1 gesetzt wird, werden die `loaddata` im optionalen Argument `\TLoad` nicht berücksichtigt und stattdessen die `loaddata` in den aktuellen `tooldata` verwendet.

Hinweis

Die Standardfunktion für den Umgang mit Nutzlasten ist die Instruktion `GripLoad`. Deshalb beträgt der Standardwert für den Systemparameter `ModalPayLoadMode` 1.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Siehe die Instruktion `MoveL` für weitere Informationen über lineare Bewegung.

Mit der Instruktion `TriggLIOs` können 1-50 unterschiedliche Triggeraktivitäten für E/A-Signale auf einer Bahn von A nach B eingerichtet werden. Es können digitale Ausgangssignale, digitale Gruppenausgangssignale und analoge Ausgangssignale verwendet werden Die Triggerbedingungen werden entweder in einer bestimmten Entfernung vor dem Endpunkt der Instruktion oder in einer bestimmten Entfernung nach dem Anfangspunkt der Instruktion erfüllt.

Für die Instruktion müssen das Argument `TriggData1`, `TriggData2` oder `TriggData3` oder alle drei Argumente verwendet werden. Die Verwendung anderer Triggerargumente ist optional. Um die Verwendung eines Triggers zu verhindern, kann die Komponente `used` in der Datenfeldkomponente der Datentypen `triggios`/`triggstrgo`/`triggiosdnum` auf FALSE gesetzt werden. Wenn keine Datenfeldkomponente verwendet wird, entspricht das Verhalten der `TriggLIOs`-Instruktion der `MoveL`-Instruktion und es werden keine E/A-Aktivitäten ausgeführt.

Bei der Vorwärtsabarbeitung des Programms werden die E/A-Aktivitäten ausgeführt. Bei der schrittweisen Rückwärtsabarbeitung werden keine E/A-Aktivitäten ausgeführt.

Wenn die Komponente `EquipLag` im `TriggData1`-, `TriggData2` oder `TriggData3`-Argument auf einen negativen Zeitwert (Verzögerung) gesetzt wird, kann das E/A-Signal hinter dem Zielpunkt (`ToPoint`) gesetzt werden.

Bei Verwendung des Arguments `TriggData2` or `TriggData3` können Werte bis 4294967295 verwendet werden. Dies ist der maximal zulässige Wert für eine Gruppe digitaler Signale (ein Gruppensignal im System darf maximal 32 Signale enthalten).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_GO_LIM`	Das programmierte `setvalue`-Argument für das angegebene digitale Gruppenausgangssignal `signalname` liegt außerhalb des zulässigen Bereichs. (Deklariert in `TriggData1`, `TriggData2` oder `TriggData3`.)
`ERR_AO_LIM`	Das programmierte `setvalue`-Argument für das angegebene analoge Ausgangssignal `signalname` liegt außerhalb des zulässigen Bereichs. (Deklariert in `TriggData1` oder `TriggData3`.)
`ERR_CONC_MAX`	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument `\Conc` verwenden, wurde überschritten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.309 TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen
RobotWare - OS
Fortsetzung

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `TriggLIOs` sind unten aufgeführt.

Beispiel 1:

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=mytriggios, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das digitale Gruppenausgangssignal `go1` wird in einer Entfernung von 3 mm von `p1` auf den Wert 55 gesetzt. Das analoge Ausgangssignal wird in einer Entfernung von 15 mm von `p1` auf den Wert 10 gesetzt. Das digitale Ausgangssignal do1 wird in einer Entfernung von 3 mm von ToPoint `p2` gesetzt.

Beispiel 2:

```
VAR triggios mytriggios{3}:= [[TRUE, 3, TRUE, 0, "go1", 55, 0],
     [TRUE, 15, TRUE, 0, "ao1", 10, 0], [TRUE, 3, FALSE, 0, "do1",
     1, 0]];
VAR triggstrgo mytriggstrgo{3}:= [[TRUE, 3, TRUE, 0, "go2", "1",
     0], [TRUE, 15, TRUE, 0, "go2", "800000", 0], [TRUE, 4, FALSE,
     0, "go2", "4294967295", 0]];
VAR triggiosdnum mytriggiosdnum{3}:= [[TRUE, 10, TRUE, 0, "go3",
     4294967295, 0], [TRUE, 10, TRUE, 0, "ao2", 5, 0], [TRUE, 10,
     TRUE, 0, "do2", 1, 0]];
...
MoveL p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=mytriggios \TriggData2:=
     mytriggstrgo \TriggData3:=mytriggiosdnum, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das digitale Gruppenausgangssignal `go1` wird in einer Entfernung von 3 mm von `p1` auf den Wert 55 gesetzt. Das analoge Ausgangssignal `ao1` wird in einer Entfernung von 15 mm von `p1` auf den Wert 10 gesetzt. Das digitale Ausgangssignal `do1` wird in einer Entfernung von 3 mm von `ToPoint p2` gesetzt. Diese Positionsereignisse werden von der Variablen `mytriggios` festgelegt. Die Variable `mytriggstrgo` legt fest, dass Positionsereignisse in einer Entfernung von 3 und 15 mm von `p1` eintreten. Zunächst wird das Signal `go2` auf 1 und dann auf 800000 gesetzt. Das Signal wird in einer Entfernung von 4 mm vom `ToPoint p2` auf 4294967295 gesetzt. Dies ist der Höchstwert für ein digitales Ausgangssignal mit 32 Bit. Die Variable `mytriggiosdnum` legt fest, dass drei Positionsereignisse in einer Entfernung von 10 mm von `p1` eintreten. Zunächst wird das Signal `go3` auf 4294967295 gesetzt, dann wird `ao2` auf 5 gesetzt und schließlich wird `do2` auf 1 gesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Wenn der aktuelle Startpunkt vom üblichen Startpunkt so abweicht, dass die gesamte Bewegungslänge der Instruktion `TriggLIOs` kürzer als normal ist (z. B. am Start von `TriggLIOs` mit der Roboterposition am Endpunkt), sind eventuell einige oder alle Triggerbedingungen sofort und an derselben Position erfüllt. In solchen Fällen ist die Abfolge der Triggeraktivitäten undefiniert. Die Logik im Anwenderprogramm darf für eine unvollständige Bewegung nicht auf einer normalen Abfolge von Triggeraktivitäten basieren.

Die Begrenzung der Triggeranzahl in der Instruktion `TriggLIOs` ist 50 für jede programmierte Instruktion. Wenn Trigger bei einer geringeren Entfernung auftreten, könnte es sein, dass das System diese nicht behandelt. Das hängt von der Art ab, in der die Bewegung durchgeführt wurde, von der benutzten TCP Geschwindigkeit, und davon, wie nah die Trigger programmiert sind. Diese Einschränkungen existieren, aber es ist schwer vorauszusagen, wann solche Probleme auftreten.

`TriggLIOs` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Begrenzungen bezüglich der Genauigkeit

Abstandsabhängige E/A-Ereignisse sind für Fly-By-Punkte (Zonenbahn) gedacht. Die Verwendung von Stopppunkten führt zu einer geringeren Genauigkeit als unten angegeben.

Hinsichtlich der Genauigkeit für E/A-Ereignisse mit Abstand und Fly-By-Punkten (Bahnverlauf) gilt Folgendes, wenn ein digitaler Ausgang im angegebenen Abstand vom Start- oder Endpunkt in der Instruktion `TriggLIOs` oder `TriggCIOs` gesetzt wird:

- Die nachstehend angegebene Geschwindigkeit gilt bei Verwendung eines positiven `equiplag`[I] der weniger als 40 ms ist, was der Verzögerung im Roboterservo ohne Änderung des Systemparameters *Event Preset Time* entspricht. Die Verzögerung kann je nach Robotertyp variieren.
- Die nachfolgend angegebene Genauigkeit gilt bei Verwendung eines positiven `equiplag`, der weniger als den konfigurierten *Event Preset Time* in den Systemparametern entspricht.
- Die unten angegebene Genauigkeit gilt nicht für einen positiven `equiplag`, der größer als der in den Systemparametern konfigurierte *Event Preset Time* ist. In diesem Fall wird eine Näherungsmethode verwendet, bei der die dynamischen Grenzen des Roboters nicht berücksichtigt werden. Dann muss `SingArea \Wrist` verwendet werden, um eine akzeptable Genauigkeit zu erzielen.
- Die nachstehend angegebene Geschwindigkeit gilt bei Verwendung eines negativen `equiplag`.

I `equiplag` ist ein Datenobjekt des Datentyps `triggios`

Die typischen absoluten Genauigkeitswerte zur Einstellung digitaler Ausgänge ist: ±5 ms.

Die typischen wiederholten Genauigkeitswerte zur Einstellung digitaler Ausgänge ist: ±2 ms.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.309 TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen

RobotWare - OS

Fortsetzung

Syntax

```
TriggLIOs
  ['\' Conc ',']
  [ToPoint ':='] <expression (IN) of robtarget >
  ['\' ID ':=' <expression (IN) of identno>]','
  [Speed ':='] <expression (IN) of speeddata>
  ['\' T ':=' <expression (IN) of num>]','
  ['\' TriggData1 ':='] <array {*} (VAR) of triggios>
  ['\' TriggData2 ':='] <array {*} (VAR) of triggstrgo>
  ['\' TriggData3 ':='] <array {*} (VAR) of triggiosdnum>','
  [Zone ':='] <expression (IN) of zonedata>
  ['\' Inpos ':=' <expression (IN) of stoppointdata>]','
  [Tool ':='] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>]
  ['\' Corr]
  ['\' TLoad ':=' <persistent (PERS) of loaddata>]';'
```

Weitere Informationen

Information über	Siehe
Robotergelenkbewegungen mit E/A-Ereignissen	*TriggJIOs – Robotergelenkbewegungen mit E/A-Ereignissen auf Seite 959*
Speicherung von Triggerbedingungen und Triggeraktivitäten	*triggios – Positioniereereignisse, trigg auf Seite 1835*
Speicherung von Triggerbedingungen und Triggeraktivitäten für digitale Signalgruppen, die aus 32 Signalen bestehen	*triggstrgo - Positioning events, trigg auf Seite 1843*
Speicherung von Triggerbedingungen und Triggeraktivitäten	*triggiosdnum – Positioniereereignisse, trigg auf Seite 1838*
Zuweisung von Ereignisobjekten	*Technisches Referenzhandbuch - Systemparameter*
Lineare Bewegung	*Technisches Referenzhandbuch - RAPID Overview*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Definition der Last	*loaddata - Lastdaten auf Seite 1723*
Beispiele für die Verwendung von `TLoad`, Gesamtlast.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*
Definieren der Nutzlast des Roboters	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
LoadIdentify, Serviceroutine zur Lastidentifikation	*Bedienungsanleitung - IRC5 mit FlexPendant*
Systemeingangssignal *SimMode* für den Betrieb des Roboters im Simulationsmodus ohne Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast	*Technisches Referenzhandbuch - Systemparameter*
Path Offset	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn

Verwendung

`TriggRampAO`(*Trigg Ramp Analog Output*) wird verwendet, um Bedingungen und Aktionen für Anlauframpen oder Auslauframpen eines analogen Ausgangssignalwerts an einer festen Position auf der Bewegungsbahn des Roboters zu definieren. Dabei kann die Verzögerung von externen Einrichtungen kompensiert werden.

Die definierten Daten können in einer oder mehreren nachfolgenden Instruktionen `TriggL`, `TriggC` oder `TriggJ` verwendet werden. Außer in diesen Instruktionen kann `TriggRampAO` auch in den Instruktionen `CapL` oder `CapC` verwendet werden.

Der Typ der mit derselben `TriggL/C/J`-Instruktion verbundenen Auslöseaktionen kann `TriggRampAO` oder eine der Instruktionen `TriggIO`, `TriggEquip`, `TriggSpeed`, `TriggInt` oder `TriggCheckIO` sein. Jede Art von Kombination ist zulässig, doch es darf nur eine `TriggSpeed`-Aktion für dasselbe Signal in derselben `TriggL/C/J`-Instruktion verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TriggRampAO`:

Siehe auch *Weitere Beispiele auf Seite 980*.

Beispiel 1:

```
VAR triggdata ramp_up;
...
TriggRampAO ramp_up, 0 \Start, 0.1, aolaser1, 8, 15;
MoveL p1, v200, z10, gun1;
TriggL p2, v200, ramp_up, z10, gun1;
```

Der aktuelle logische Wert des analogen Signals `aolaser1` wird auf den neuen Wert 8 erhöht, wenn der TCP des Werkzeugs `gun1` 0,1 s vom Mittelpunkt der Zonenbahn bei p1 entfernt ist. Die gesamte Auslauframpe erfolgt während einer Roboterbewegung von 15 mm.

Beispiel 2:

```
VAR triggdata ramp_down;
...
TriggRampAO ramp_down, 15, 0.1, aolaser1, 2, 10;
MoveL p3, v200, z10, gun1;
TriggL p4, v200, ramp_down, z10, gun1;
```

Der aktuelle logische Wert des analogen Signals `aolaser1` wird auf den neuen Wert 2 erhöht, wenn der TCP des Werkzeugs `gun1` 0,1 s vom Mittelpunkt der Zonenbahn bei p4 entfernt ist. Die gesamte Auslauframpe erfolgt während einer Roboterbewegung von 10 mm.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

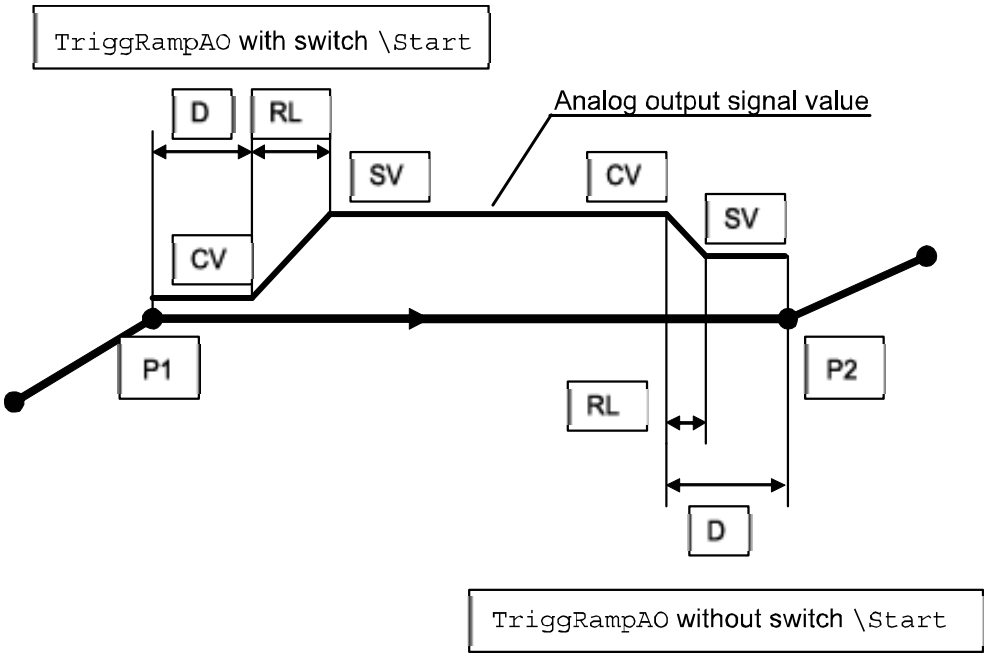
1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn

RobotWare - OS

Fortsetzung

Argumente

```
TriggRampAO TriggData Distance [\Start] | [\Next] EquipLag AOutput
     SetValue RampLength [\Time] [\Inhib] [\InhibSetValue] [\Mode]
```



xx0600003433

D	Parameter `Distance`
RL	Parameter `RampLength`
CV	Aktueller Wert des analogen Signals
SV	Parameter `SetValue` für den Wert des analogen Signals
P1	`ToPoint` für die vorherige Bewegungsinstruktion
P2	`ToPoint` für die gegenwärtige `TrigL/C/J` -Instruktion

`TriggData`

Datentyp: `triggdata`

Variable zum Speichern der `triggdata`, die von dieser Instruktion zurückgegeben werden. Diese `triggdata` können dann in der nachfolgenden Instruktion `TriggL`, `TriggC` , `TriggJ`, `CapL` oder `CapC` verwendet werden.

`Distance`

Datentyp: `num`

Definiert den Abstand vom Mittelpunkt der Zonenbahn, an dem die Rampe des analogen Ausgangssignals beginnt.

Angegeben als Abstand in mm (positiver Wert) vom Endpunkt (`ToPoint`) der Bewegungsbahn (gültig, wenn das Argument `\Start` nicht gesetzt wurde).

Ausführliche Anleitungen finden Sie unter *Programmabarbeitung auf Seite 978*.

`[\Start]`

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn

Wird verwendet, wenn der Abstand für Argument `Distance` ab dem Startpunkt (vor `ToPoint`) anstelle des Endpunkts der Bewegung gemessen wird.

`[ \Next ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für das Argument `Distance` auf dem Weg zum nächsten programmierten Punkt ist. Wenn die `Distance` länger ist als die Entfernung zum nächsten Feinpunkt, wird das Ereignis am Feinpunkt ausgeführt.

`EquipLag`

Equipment Lag

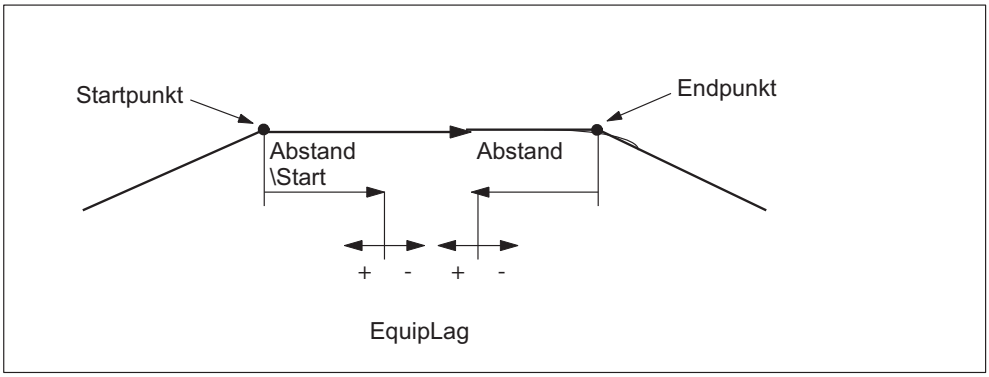
Datentyp: `num`

Gibt die Verzögerung der externen Einrichtung in s an.

Verwenden Sie zur Kompensierung der Verzögerung einen positiven Argumentwert. Ein positiver Argumentwert bedeutet, dass das Robotersystem den Beginn der Rampe des `AO`-Signals zur angegebenen Zeit ausführt, bevor der `TCP` den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung physisch erreicht.

Ein negativer Argumentwert bedeutet, dass das Robotersystem den Beginn der Rampe des `AO`-Signals zur angegebenen Zeit setzt, nachdem der `TCP` den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung physisch passiert hat.

Die Abbildung zeigt die Verwendung von Argument `EquipLag`.



xx0500002262

`AOutput`

Analog Output

Datentyp: `signalao`

Name des analogen Ausgangssignals

`SetValue`

Datentyp: `num`

Der Wert, auf den das analoge Ausgangssignal herauf- oder heruntergesetzt werden soll (muss im zulässigen logischen Wertebereich für das Signal liegen). Die Rampe wird mit dem aktuellen Wert des analogen Ausgangssignals begonnen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn
RobotWare - OS
Fortsetzung

`RampLength`

Datentyp: `num`

Die Rampenlänge in mm auf der `TCP`-Bewegungsbahn.

`[\Time]`

Datentyp: `switch`

Wird sie verwendet, dann gibt die `RampLength` die Rampenzeit in Sekunden und nicht als Längenmaß an.

Muss verwendet werden, wenn ein nachfolgendes `TriggL, TriggC` oder `TriggJ` angibt, dass die gesamte Bewegung mit Zeitwerten (Argument `\T`) anstelle von Geschwindigkeitswerten ausgeführt werden soll.

`[ \Inhib ]`

Inhibit

Datentyp: `bool`

Der Name eines Flags für persistente Variablen, das das Setzen des Signals zur Laufzeit verhindert.

Wenn das optionale Argument verwendet wird und der aktuelle Wert des angegebenen Flags zu dem Zeitpunkt, an dem das Signal gesetzt werden soll, TRUE ist, wird das angegebene Signal (`AOutput`) nicht auf den angegebenen Wert sondern auf 0 gesetzt.

`[ \InhibSetValue ]`

InhibitSetValue

Datentyp: `bool, num or dnum`

Der Name einer persistenten Variablen des Datentyps `bool`, `num` oder `dnum` oder jedes Alias dieser drei Basisdatentypen.

Das optionale Argument kann nur gemeinsam mit dem optionalen Argument `Inhib` verwendet werden.

Wenn dieses optionale Argument verwendet wird und der Wert des persistenten variablen Flags, das im optionalen Argument `Inhib` verwendet wird, an der Positionszeit für die Signaleinstellung TRUE ist, wird der Wert der persistenten Variable gelesen, die im optionalen Argument `InhibSetValue` verwendet wird. Dieser Wert wird zur Einstellung des Signals `AOutput` verwendet.

Bei Verwendung einer persistenten booleschen Variable wird der Wert TRUE in den Wert 1 und FALSE in den Wert 0 übertragen.

`[ \Mode ]`

Datentyp: `triggmode`

Wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

Programmabarbeitung

Beim Ausführen der Instruktion `TriggRampAO` wird die Triggerbedingung in der angegebenen Variablen für Argument `TriggData` gespeichert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

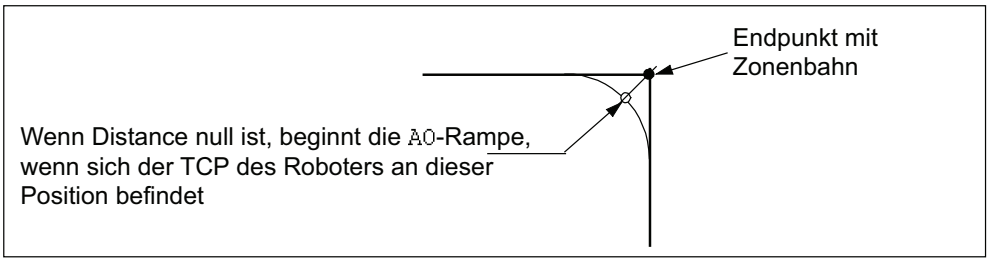
1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn

Wenn danach eine der Instruktionen `TriggL`, `TriggC` oder `TriggJ` abgearbeitet wird, gilt hinsichtlich der Definitionen in `TriggRampAO` Folgendes:

Die Tabelle beschreibt den im Argument `Distance` angegebenen Abstand:

Lineare Bewegung	Der geradlinige Abstand
Kreisbewegung	Die Länge des Kreisbogens
Nicht lineare Bewegung	Die ungefähre Bogenlänge entlang der Bahn (für angemessene Genauigkeit sollte der Abstand nicht die Hälfte der Bogenlänge überschreiten.)

Die Abbildung zeigt eine `AO`-Rampe auf einer Zonenbahn.



xx0600003439

Eigenschaften der Programmabarbeitung von mit `TriggL/C/J` verbundenem `TriggRampAO`:

- Die Rampe von `AO` beginnt, wenn der Roboter den angegebenen `Distance`-Punkt auf der Roboterbahn erreicht (mit Kompensierung des angegebenen `EquipLag`).
- Die Rampenfunktion wird während der Zeitspanne ausgeführt, die anhand der angegebenen `RampLength` und der programmierten `TCP`-Geschwindigkeit berechnet wird. Bei der Berechnung werden `VelSet`, manueller Geschwindigkeitsoverride und maximal 250 mm/s im Modus `MAN`, jedoch keine anderen Geschwindigkeitsbeschränkungen berücksichtigt.
- Der `AO`-Signalwert ab dem Startwert (aktuelle Ablesung) wird alle 10 ms auf den angegebenen `SetValue` aktualisiert. Dies führt zu einer treppenförmigen Kurve. Wenn die berechnete oder angegebene Rampenzeit größer als 0,5 s ist, verringert sich die Rampenfrequenz:
  - <= 0,5 s ergibt maximal 50 Stufen pro 10 ms
  - <= 1 s ergibt maximal 50 Stufen pro 20 ms
  - <= 1,5 s ergibt maximal 50 Stufen pro 30 ms usw.

Die `TriggRampAO`-Aktion erfolgt im `FWD`-Schrittmodus, doch nicht im `BWD`-Schrittmodus.

Bei jedem Typ von Stopp (ProgStop, Not-Halt …), wenn die Rampenfunktion für diesen Fall aktiviert ist:

- Bei einer Anlauframpe wird der `AO` sofort auf den vorherigen Wert gesetzt.
- Bei einer Auslauframpe wird der `AO` sofort auf den neuen `SetValue` gesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte SetValue-Argument für das angegebene analoge Ausgangssignal AOutput liegt außerhalb des zulässigen Bereichs.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion TriggRampAO sind unten aufgeführt.

Beispiel 1:

```
VAR triggdata ramp_up;
VAR triggdata ramp_down;
...
TriggRampAO ramp_up, 0 \Start, 0.1, aolaser1, 8, 15;
TriggRampAO ramp_down, 15, 0.1, aolaser1, 2, 10;
MoveL p1, v200, z10, gun1;
TriggL p2, v200, ramp_up, \T2:=ramp_down, z10, gun1;
```

In diesem Beispiel werden die Anlauframpe und Auslauframpe von AO in derselben TriggL-Instruktion und auf derselben Bewegungsbahn ausgeführt. Die Ausführung kann ohne Beeinflussung durch AO-Einstellungen erfolgen, wenn die Bewegungsbahn eine ausreichende Länge aufweist.

Der aktuelle logische Wert des analogen Signals aolaser1 wird auf den neuen Wert 8 erhöht, wenn der TCP des Werkzeugs gun1 0,1 s vom Mittelpunkt der Zonenbahn bei p1entfernt ist. Die gesamte Auslauframpe erfolgt während einer Roboterbewegung von 15 mm.

Das Analogsignal aolaser1 beginnt das Auslaufen seines logischen Werts von dem aktuellen Wert 8 auf den neuen Wert 2, wenn der TCP des Werkzeugs gun1 15 mm beträgt, plus 0,1 s vor der Mitte der Zonenbahn bei p2. Die gesamte Auslauframpe erfolgt während einer Roboterbewegung von 10 mm.

Einschränkungen

Eine geringere TCP-Geschwindigkeit auf der Zonenbahn sowie Beschleunigungs- oder Verzögerungsphasen werden im analogen Ausgangssignalwert nicht kompensiert (der AO ist nicht proportional zur TCP-Geschwindigkeit).

Nur der Startpunkt der AO-Rampe befindet sich an der angegebenen Position auf der Bahn. Die Anlauf- und Auslauframpe wird mit hoher Genauigkeit berechnet.

- Bei konstanter Geschwindigkeit ist die Abweichung für das Ende der AO-Rampe von der angegebenen Position gering.
- Während Beschleunigungs- oder Verzögerungsphasen, z. B. in der Nähe von Stopppunkten, ist die Abweichung größer.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn

- Empfehlung: Verwenden Sie Zonenbahnen vor Anlauframpen und nach Ablauframpen.

Wenn mehrere `TriggRampAO` für dasselbe analoge Ausgangssignal verwendet werden, mit derselben `TriggL/C/J` -Instruktion verbunden sind und wenn sich mehrere `RampLength` in demselben Bereich der Roboterbahn befinden, beeinflussen sich die `AO`-Einstellungen gegenseitig.

Das positionsabhängige (+/- Zeit) `AO`-Rampenereignis beginnt, wenn der vorherige `ToPoint` passiert wurde und die angegebene `Distance` vom gegenwärtigen `ToPoint` nicht innerhalb der Bewegungslänge für die gegenwärtige `TriggL/C/J`-Instruktion liegt. Das positionsabhängige (+/- Zeit) `AO`-Rampenereignis beginnt, wenn der gegenwärtige `ToPoint` passiert wurde und die angegebene `Distance` vom vorherigen `ToPoint` nicht innerhalb der Bewegungslänge für die aktuelle `TriggL/C/J`-Instruktion (mit dem Argument `\Start`) liegt.

Der Neustart der `AO`-Rampenfunktion nach einem beliebigen Typ von Stopp (ProgStop, Not-Halt ...) wird nicht unterstützt.

Beim Neustart nach einem Stromausfall startet die `TriggL/C/J`-Instruktion ab der beim Stromausfall aktuellen Position.

`TriggRampAO` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggRampAO
  [ TriggData ':=' ] < variable (VAR) of triggdata > ','
  [ Distance ':=' ] < expression (IN) of num >
  [ '\' Start ] | [ '\' Next ] ','
  [ EquipLag ':=' ] < expression (IN) of num > ','
  [ AOutput ':=' ] < variable (VAR) of signalao> ','
  [ SetValue ':=' ] < expression (IN) of num> ','
  [ RampLength ':=' ] < expression (IN) of num> ','
  [ '\' Time ]
  [ '\' Inhib' :=' < persistent (PERS) of bool> ]
  [ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
  [ '\' Mode' :=' < expression (IN) of triggmode> ] ';'
```

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von anderen Triggern	*TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.310 TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn

RobotWare - OS

Fortsetzung

Information über	Siehe
Speichern von triggdata	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *triggmode - Triggeraktionsmodus auf Seite 1840*
Setzen eines analogen Ausgangssignals	*SetAO - Ändern des Werts eines analogen Ausgangssignals auf Seite 700* *signalxx - Digitale und analoge Signale auf Seite 1791*
Konfiguration der Ereignis-Vorsetzzeit	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung

Verwendung

`TriggSpeed` wird verwendet zur Definition von Bedingungen und Aktionen zur Steuerung eines analogen Ausgangssignals mit einem Wert, der proportional zur aktuellen TCP-Geschwindigkeit ist. Der Anfang, die Skalierung und das Ende der analogen Ausgabe können bei einer festen Position/Zeit auf der Bewegungsbahn des Roboters angegeben werden. Die Verzögerung bei externen Geräten und durch Geschwindigkeitsänderungen am Roboter kann für den Anfang, die Skalierung und das Ende des Ausgangssignals zeitlich ausgeglichen werden.

Die definierten Daten können in einer oder mehreren aufeinander folgenden `TriggL`, `TriggC` oder `TriggJ` Instruktionen verwendet werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` bzw. in einem MultiMove-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

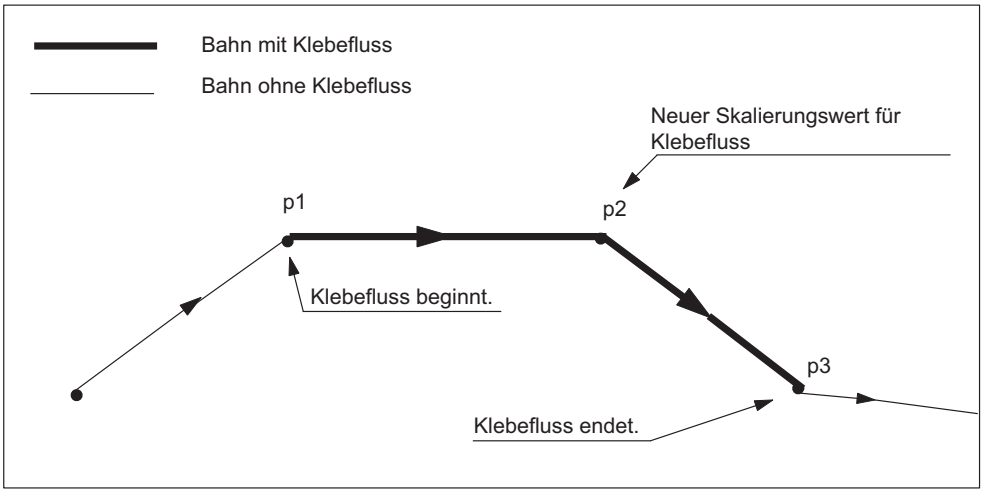
Das folgende Beispiel veranschaulicht die Instruktion `TriggSpeed`:

Siehe auch *Weitere Beispiele auf Seite 990*.

Beispiel 1:

```
VAR triggdata glueflow;
TriggSpeed glueflow, 0, 0.05, glue_ao, 0.8\DipLag:=0.04
     \ErrDO:=glue_err;
TriggL p1, v500, glueflow, z50, gun1;
TriggSpeed glueflow, 10, 0.05, glue_ao, 1;
TriggL p2, v500, glueflow, z10, gun1;
TriggSpeed glueflow, 0, 0.05, glue_ao, 0;
TriggL p3, v500, glueflow, z50, gun1;
```

Die Abbildung unten zeigt ein Beispiel der `TriggSpeed`-Sequenz.



xx0500002329

Der Klebefluss (analoger Ausgang `glue_ao`)) mit dem Skalierungswert `0.8` beginnt, wenn sich der TCP `0.05` s vor dem Punkt `p1` befindet, mit dem neuen

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung

RobotWare - OS

Fortsetzung

Klebefluss-Skalierungswert `1`, wenn sich der TCP `10` mm plus `0.05` s vor dem Punkt `p2` befindet, und der Klebefluss endet (Skalierungswert 0), wenn sich der TCP `0.05` s vor dem Punkt`p3` befindet.

Geschwindigkeitsabnahmen des Roboters werden zeitlich ausgeglichen, indem das analoge Ausgangssignal `glue_ao``0.04` s vor ihrem Auftreten beeinflusst wird.

Bei einem Überlauf des errechneten logischen analogen Ausgangswerts in `glue_ao` wird das digitale Ausgangssignal`glue_err` gesetzt. Ist der Überlauf behoben, wird `glue_err` zurückgesetzt.

Argumente

```
TriggSpeed TriggData Distance [\Start] | [\Next] ScaleLag AOp
     ScaleValue [\DipLag] [\ErrDO] [\Inhib] [\InhibSetValue]
     [\Mode]
```

`TriggData`

Datentyp: `triggdata`

Variable zum Speichern der `triggdata` , die von dieser Instruktion zurückgegeben werden. Diese `triggdata` werden dann in der nachfolgenden Instruktion `TriggL`, `TriggC` oder `TriggJ` verwendet.

`Distance`

Datentyp: `num`

Definiert die Position auf der Bahn, an der die Änderung des analogen Ausgangswerts vorgenommen wird.

Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn (gültig, wenn das Argument `\Start` nicht gesetzt wurde).

Ausführliche Anleitungen finden Sie unter *Programmabarbeitung auf Seite 988*.

`[ \Start ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für Argument `Distance` ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`[ \Next ]`

Datentyp: `switch`

Wird verwendet, wenn der Abstand für das Argument `Distance` vorwärts zum nächsten programmierten Punkt ist.

Dieses Argument kann verwendet werden, um die Änderung des Analogausgangswertes nach dem Endpunkt der Bewegungsbahn zu verschieben. Es sollte nur verwendet werden, wenn die Signaländerung so abgestimmt werden

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

soll, dass sie kurz nach dem `ToPoint` der verwendeten Bewegungsinstruktion (`TriggL`, `TriggC` oder `TriggJ`-Instruktionen) liegt.

> **Hinweis**
>
> Der Programmierer muss vorsichtig vorgehen, wenn er das `\Next`-Argument verwendet. Je nach verwendetem Abstand und Argumenten in der folgenden Auslösebedingung kann es zu Aktionsüberschneidungen kommen, und das Ergebnis ist möglicherweise nicht wie erwartet.

`ScaleLag`

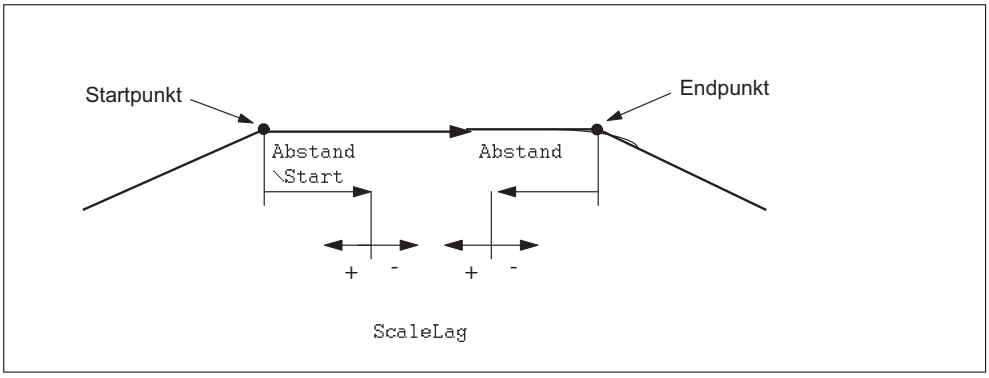
Datentyp: `num`

Gibt die zeitliche Verzögerung in den externen Geräten in s an (positiver Wert), für die die Änderung des analogen Ausgangswerts (Anfang, Skalierung, Ende) vorgenommen werden soll.

Für die Kompensierung der Verzögerung in externen Geräten bedeutet dieser Argumentwert, dass das analoge Ausgangssignal vom Roboter zur angegebenen Zeit gesetzt wird, bevor der TCP physisch den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung erreicht hat.

Das Argument kann auch verwendet werden, um das analoge Ausgangssignal über den Endpunkt hinaus zu verlängern. Legen Sie die Zeit, in der der Roboter das analoge Ausgangssignal beibehalten soll, in Sekunden fest. Geben Sie der Zeit ein negatives Vorzeichen. Die Grenze liegt bei -0,10 Sekunden.

Die Abbildung unten zeigt die Verwendung des Arguments `ScaleLag`



xx0500002330

`AOp`

Analog Output

Datentyp: `signalao`

Name des analogen Ausgangssignals

`ScaleValue`

Datentyp: `num`

Skalierungswert des analogen Ausgangssignals

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung

RobotWare - OS

Fortsetzung

Der physikalische Ausgangswert des analogen Signals wird vom Roboter berechnet:

- Logischer Ausgangswert = Skalierungswert * Tatsächliche TCP-Geschwindigkeit in mm/s
- Physikalischer Ausgangswert = Laut Definition in der Konfiguration für aktuelle analoge Ausgangssignale unter „Logischer Ausgangswert als Eingang“

`[ \DipLag ]`

Datentyp: `num`

Geben Sie die zeitliche Verzögerung in den externen Geräten in s an (positiver Wert), wenn der analoge Ausgangswert wegen einer Verringerung der Robotergeschwindigkeit geändert wird.

Für die Kompensation der Verzögerung von externen Geräten bedeutet dieser Argumentwert, dass das analoge Ausgangssignal vom Roboter zum angegebenen Zeitpunkt vor der Abnahme der TCP-Geschwindigkeit gesetzt wird.

Hinweis

Dieses Argument kann vom Roboter nur für die erste `TriggSpeed`-Instruktion (zusammen mit einer der Instruktionen `TriggL`, `TriggC` oder `TriggJ`) aus einer Sequenz von `TriggSpeed`-Instruktionen verwendet werden. Der erste angegebene Argumentwert gilt für alle folgenden `TriggSpeed`-Instruktionen in der Sequenz.

`[ \ErrDO ]`

Error Digital Output

Datentyp: `signaldo`

Name des digitalen Ausgangssignals, das den Überlauf des analogen Signals meldet

Wenn die Berechnung des logischen analogen Ausgangswerts für das Signalargument `AOp` während einer Bewegung zu einem Überlauf wegen Geschwindigkeitsüberschreitung führt, wird dieses Signal gesetzt und der physikalische analoge Ausgangswert auf den Höchstwert reduziert. Ist der Überlauf behoben, wird das Signal zurückgesetzt.

Hinweis

Dieses Argument kann vom Roboter nur für die erste `TriggSpeed`-Instruktion (zusammen mit einer der Instruktionen `TriggL`, `TriggC` oder `TriggJ`) in einer Sequenz von mehreren `TriggSpeed`-Instruktionen verwendet werden. Der erste angegebene Argumentwert gilt für alle folgenden `TriggSpeed`-Instruktionen in der Sequenz.

`[ \Inhib ]`

Inhibit

Datentyp: `bool`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Name eines Speichernden-Flags, das das Setzen des Signals bei der Abarbeitung verhindert.

Wenn das optionale Argument verwendet wird und der aktuelle Wert des angegebenen Flags zu dem Zeitpunkt, an dem das analoge Signal gesetzt werden soll, `TRUE` lautet, wird das angegebene Signal `AOp` statt auf den berechneten Wert auf 0 gesetzt.

Hinweis

Dieses Argument kann vom Roboter nur für die erste `TriggSpeed`-Instruktion (zusammen mit einer der Instruktionen `TriggL`, `TriggC` oder `TriggJ`) in einer Sequenz von mehreren `TriggSpeed`-Instruktionen verwendet werden. Der erste angegebene Argumentwert gilt für alle folgenden `TriggSpeed`-Instruktionen in der Sequenz.

`[ \InhibSetValue ]`

InhibitSetValue

Datentyp: `bool, num or dnum`

Der Name einer persistenten Variablen des Datentyps `bool`, `num` oder `dnum` oder jedes Alias dieser drei Basisdatentypen.

Das optionale Argument kann nur gemeinsam mit dem optionalen Argument `Inhib` verwendet werden.

Wenn dieses optionale Argument verwendet wird und der Wert des persistenten variablen Flags, das im optionalen Argument `Inhib` verwendet wird, an der Positionszeit für die Signaleinstellung TRUE ist, wird der Wert der persistenten Variable gelesen, die im optionalen Argument `InhibSetValue` verwendet wird. Dieser Wert wird zur Einstellung des Signals `AOp` verwendet.

Bei Verwendung einer persistenten booleschen Variable wird der Wert TRUE in den Wert 1 und FALSE in den Wert 0 übertragen.

Hinweis

Dieses Argument kann vom Roboter nur für die erste `TriggSpeed`-Instruktion (zusammen mit einer der Instruktionen `TriggL`, `TriggC` oder `TriggJ`) in einer Sequenz von mehreren `TriggSpeed`-Instruktionen verwendet werden. Der erste angegebene Argumentwert gilt für alle folgenden `TriggSpeed`-Instruktionen in der Sequenz.

`[ \Mode ]`

Datentyp: `triggmode`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung

RobotWare - OS

Fortsetzung

Wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

> **Hinweis**
>
> Dieses Argument kann vom Roboter nur für die erste `TriggSpeed`-Instruktion (zusammen mit einer der Instruktionen `TriggL`, `TriggC` oder `TriggJ`) in einer Sequenz von mehreren `TriggSpeed`-Instruktionen verwendet werden. Der erste angegebene Argumentwert gilt für alle folgenden `TriggSpeed`-Instruktionen in der Sequenz.

Programmabarbeitung

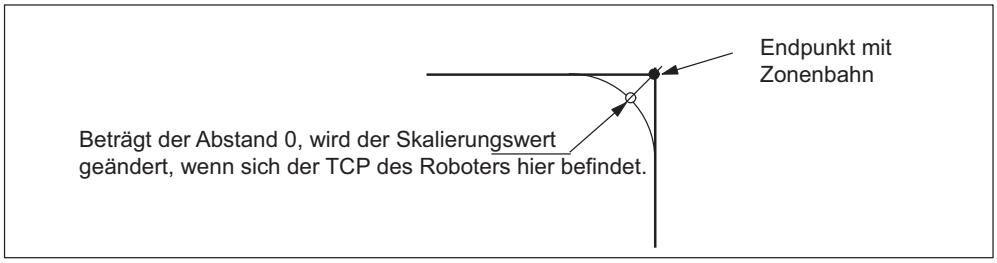
Beim Ausführen der Instruktion `TriggSpeed` wird die Triggerbedingung in der angegebenen Variablen für das Argument `TriggData` gespeichert.

Wenn danach eine der Instruktionen `TriggL`, `TriggC` oder `TriggJ` ausgeführt wird, gilt hinsichtlich der Definitionen in `TriggSpeed` Folgendes:

Die Tabelle unten beschreibt den im Argument `Distance` angegebenen Abstand:

Lineare Bewegung	Der geradlinige Abstand
Kreisbewegung	Die Länge des Kreisbogens
Nicht lineare Bewegung	Die ungefähre Bogenlänge entlang der Bahn (für angemessene Genauigkeit sollte der Abstand nicht die Hälfte der Bogenlänge überschreiten.)

Die Abbildung unten zeigt den zeit- und positionsabhängigen Skalierungswert auf einer Zonenbahn.



xx0500002331

Das positions- und zeitabhängige Skalierungswertereignis wird generiert, wenn der Startpunkt (Endpunkt) passiert wird, falls der angegebene Abstand vom Endpunkt (Startpunkt) nicht innerhalb der Bewegungslänge der aktuellen Instruktion liegt (`TriggL`, `TriggC` oder `TriggJ`).

Die erste `TriggSpeed` -Instruktion, die von einer `TriggL`, `TriggC` oder `TriggJ`-Instruktion verwendet wird, erstellt im System einen Prozess, der denselben Namen wie das analoge Ausgangssignal aufweist. Für alle folgenden `TriggL`, `TriggC` oder `TriggJ`-Instruktionen, die mithilfe der `TriggSpeed` -Instruktion auf denselben Signalnamen und dieselbe Einstellung verweisen, wird der gleiche Prozess verwendet.

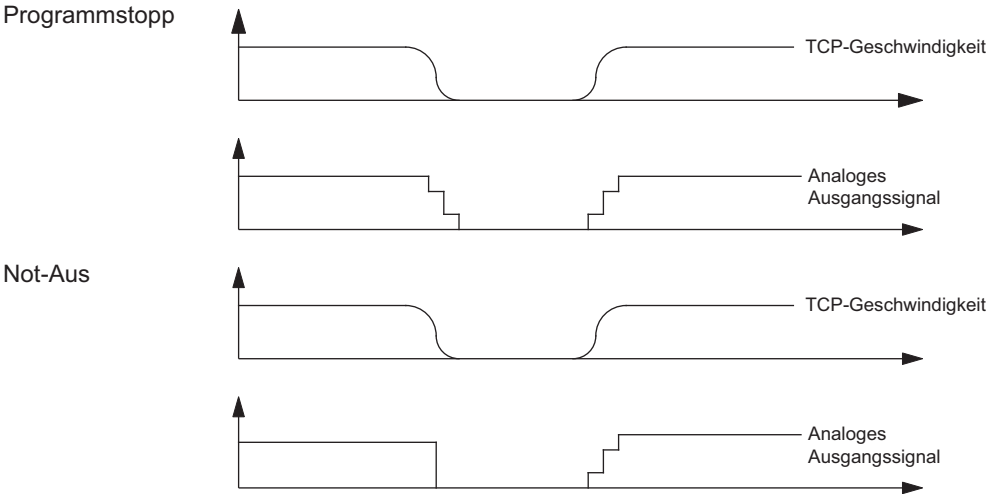
Der Prozess setzt das analoge Ausgangssignal bei einem Programmstopp oder Not-Halt unmittelbar auf 0. Im Falle eines Programmstopps bleibt das analoge Ausgangssignal bis zum Stillstand des Roboters proportional zur

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fortsetzung

TCP-Geschwindigkeit. Der Prozess bleibt aktiv und bereit für einen Neustart. Beim Neustart des Roboters ist das Signal von Anfang an proportional zur TCP-Geschwindigkeit.



xx0500002332

Der Prozess wird nach einem Skalierungsereignis mit dem Wert 0 ungültig, wenn zu diesem Zeitpunkt nicht die Instruktion `TriggL`, `TriggC` oder `TriggJ` in der Warteschlange vorhanden ist.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

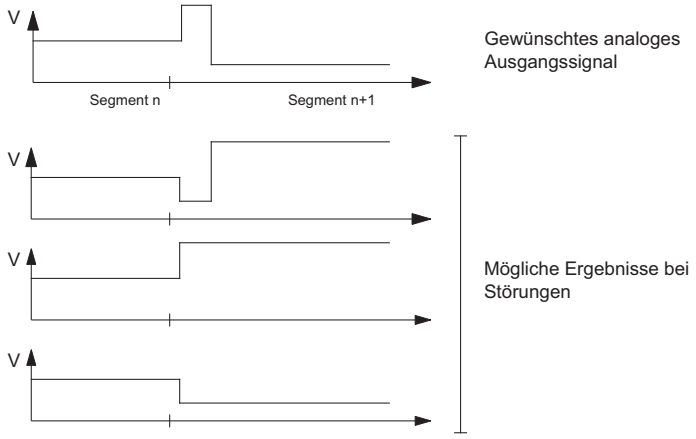
Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.

Es sind zwei aufeinanderfolgende Bewegungsfolgen mit `TriggL`/`TriggSpeed`-Instruktionen angegeben. Durch einen negativen Wert im Parameter `ScaleLag` kann das Skalierungsereignis aus dem ersten Segment an den Anfang des zweiten Segments verschoben werden. Wenn am Anfang des

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

zweiten Segments eine Skalierung vorhanden ist, kann nicht gesteuert werden, ob sich die beiden Skalierungen störend beeinflussen.



xx0500002334

Weitere Beispiele

Weitere Beispiele für die Instruktion `TriggSpeed` sind unten aufgeführt.

Beispiel 1:

```
VAR triggdata flow;
TriggSpeed flow, 10 \Start, 0.05, flowsignal, 0.5 \DipLag:=0.03;
MoveJ p1, v1000, z50, tool1;
TriggL p2, v500, flow, z50, tool1;
```

Das analoge Ausgangssignal `flowsignal` wird auf einen logischen Wert = (`0.5` * tatsächliche TCP-Geschwindigkeit in mm/s) `0.05` s gesetzt, bevor der TCP einen Punkt überschreitet, der sich `10` mm hinter dem Startpunkt `p` befindet. Der Ausgangswert wird so angepasst, dass er proportional zur tatsächlichen TCP-Geschwindigkeit bei der Bewegung nach `p2` ist.

```
...
TriggL p3, v500, flow, z10, tool1;
```

Der Roboter bewegt sich weiterhin mit dem Wert des analogen Ausgangs, der proportional zur tatsächlichen TCP-Geschwindigkeit ist, von `p2` nach `p3`, wobei der analoge Ausgangssignalwert proportional zur aktuellen TCP-Geschwindigkeit ist. Der analoge Ausgangswert wird zum Zeitpunkt `0.03` s vor der Verringerung der Robotergeschwindigkeit beim Passieren der Zonenbahn `z10` herabgesetzt.

Beispiel 2:

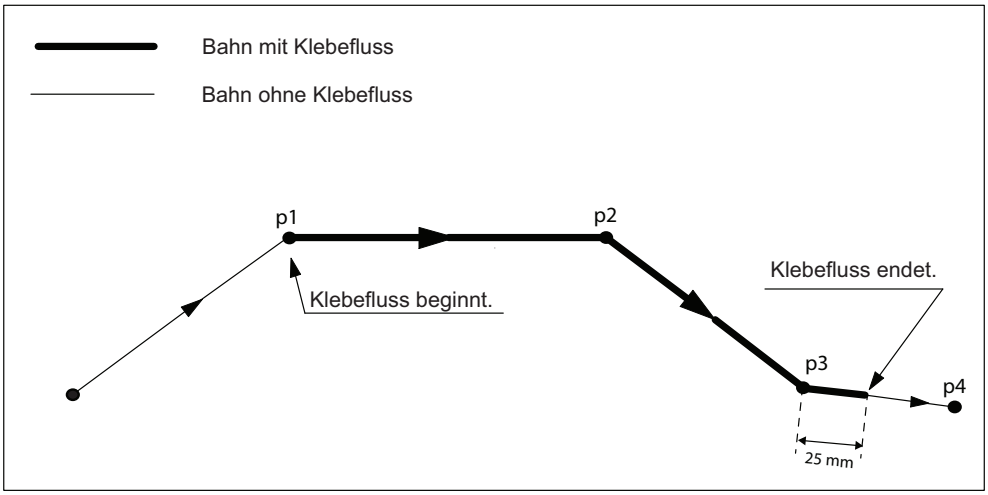
```
VAR triggdata glueflow;
VAR triggdata glueflowend;
TriggSpeed glueflow, 0, 0.05, glue_ao, 1;
TriggSpeed glueflowend, 25 \Next, 0, glue_ao, 0;
TriggL p1, v500, glueflow, z50, gun1;
TriggL p2, v500, glueflow, z50, gun1;
TriggL p3, v500, glueflowend, z50, gun1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
MoveL p4, v500, z50, gun1;
```

Die Abbildung unten zeigt ein Beispiel der `TriggSpeed`-Sequenz und die Verwendung des `\Next`-Arguments.



xx1800000010

Der Leimdurchfluss (Analogausgang `glue_ao`) mit Skalenwert `0.8` beginnt, wenn TCP `0.05` s vor Punkt `p1` liegt. Der Leimdurchfluss endet (Skalenwert 0), wenn TCP `25` mm nach dem Punkt `p3` liegt.

Verwandte Systemparameter

Der Systemparameter *Event Preset Time* wird verwendet, um den Roboter so zu verzögern, dass externe Einrichtungen vor dem Erreichen der jeweiligen Roboterposition aktiviert/gesteuert werden können.

Die Tabelle unten zeigt die Empfehlungen für die Einstellung des Systemparameters *Event Preset Time* für einen typischen Servo Lag von 0,060 s.

`ScaleLag`	`DipLag`	Erforderliche *Event Preset Time* zur Vermeidung von Laufzeitfehlern bei der Abarbeitung	Erforderliche *Event Preset Time* für höchste Genauigkeit
`ScaleLag > DipLag`	Immer	`DipLag`, wenn `DipLag > ServoLag`	`ScaleLag` in s plus 0,090 s
`ScaleLag < DipLag`	`DipLag < Servo Lag`	`- " -`	0.090 s
`- " -`	`DipLag > Servo Lag`	`- " -`	`DipLag` in s plus 0,030 s

Einschränkungen

Die Einschränkungen für die Instruktion `TriggSpeed` sind unten aufgeführt.

Genauigkeit eines positions- und zeitabhängigen Skalierungswertereignisses

Typische absolute Genauigkeit für Skalierungswertereignisse beträgt ±5 ms.

Typische Wiederholgenauigkeit für Skalierungswertereignisse beträgt ±2 ms.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung

RobotWare - OS

Fortsetzung

Genauigkeit der Anpassung an Abnahmen in der TCP-Geschwindigkeit (Verzögerungs-/Beschleunigungsphasen)

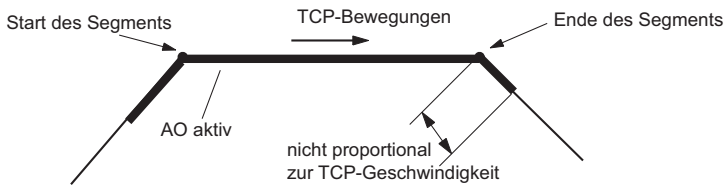
Typische absolute Genauigkeit für TCP-Geschwindigkeitsabnahmen beträgt ±5 ms.

Typische Wiederholgenauigkeit für TCP-Geschwindigkeitsabnahmen beträgt ±2 ms (der Wert hängt von der konfigurierten *Path resolution* ab).

Negative `ScaleLag`

Wenn für den Parameter `ScaleLag` ein negativer Wert verwendet wird, um die Null-Skalierung in das nächste Segment zu verschieben, wird das analoge Ausgangssignal bei einem Programmstopp nicht zurückgesetzt. Bei einem Not-Halt wird das analoge Signal immer zurückgesetzt.

Nach dem Endpunkt des Segments ist das analoge Signal nicht mehr proportional zur TCP-Geschwindigkeit.



xx0500002333

`TriggSpeed` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggSpeed
  [ TriggData ':=' ] < variable (VAR) of triggdata>','
  [ Distance' :=' ] < expression (IN) of num>
  [ '\' Start ] | [ '\' Next ] ','
  [ ScaleLag':=' ] < expression (IN) of num> ','
  [ AOp ':='] < variable (VAR) of signalao> ','
  [ ScaleValue' :=' ] < expression (IN) of num>
  [ '\' DipLag' :=' < expression (IN) of num> ]
  [ '\' ErrDO' :=' < variable (VAR ) of signaldo> ]
  [ '\' Inhib' :=' < persistent (PERS ) of bool >]
  [ '\' InhibSetValue' :=' < persistent (PERS) of anytype> ]
  [ '\' Mode' :=' < expression (IN) of triggmode> ] ';'
```

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.311 TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung

RobotWare - OS

Fortsetzung

Information über	Siehe
Definition von anderen Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923*
Speichern von triggdata	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *triggmode - Triggeraktionsmodus auf Seite 1840*
Konfiguration von Event preset time	*Technisches Referenzhandbuch - Systemparameter*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.312 TriggStopProc - Generieren von Neustartdaten für Trigg-Signale bei einem Stopp

Verwendung

Die Instruktion `TriggStopProc` erstellt einen internen Überwachungsprozess im System, um bei jedem Programmstopp (`STOP`) oder Not-Halt (`QSTOP`) im System angegebene Prozesssignale auf null zu setzen und Neustartdaten in einer angegebenen persistenten Variablen zu erzeugen.

`TriggStopProc` und der Datentyp `restartdata` dienen dem Neustart nach einem Programmstopp (`STOP`) oder Not-Halt (`QSTOP`) der eigenen Prozessinstruktionen, die in RAPID (`NOSTEPIN`-Routinen) definiert sind.

In einer anwenderdefinierten `RESTART`-Ereignisroutine können vor dem Neustart der Bewegungen die aktuellen Neustartdaten analysiert und mit der Instruktion `StepBwdPath` können Schritte rückwärts auf der Bahn ausgeführt und entsprechende Prozesssignale aktiviert werden.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in beliebigen Bewegungstasks verwendet werden.

Bei einem MultiMove-System kann immer nur ein `TriggStopProc`-Unterstützungsprozess mit dem angegebenen Schattensignalnamen (Argument `ShadowDO`) im System aktiv sein. Das heißt, dass `TriggStopProc` den Programmstopp oder das Not-Aus dort in der Programmtask überwacht, wo der Programmstopp oder das Not-Halt zuletzt ausgeführt wurde.

Argumente

```
TriggStopProc RestartRef [\DO] [\GO1] [\GO2] [\GO3] [\GO4] ShadowDO
```

`RestartRef`

Restart Reference

Datentyp: `restartdata`

Die persistente Variable, in der nach jedem Programmstopp Neustartdaten verfügbar sind.

`[\DO1]`

Digital Output 1

Datentyp: `signaldo`

Die Signalvariable für ein digitales Prozesssignal, die auf null gesetzt und nach einem Programmstopp über Neustartdaten überwacht werden soll.

`[\GO1]`

Group Output 1

Datentyp: `signalgo`

Die Signalvariable für ein digitales Gruppenprozesssignal, die auf null gesetzt und nach einem Programmstopp über Neustartdaten überwacht werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\GO2]

Group Output 2

Datentyp: signalgo

Die Signalvariable für ein digitales Gruppenprozesssignal, die auf null gesetzt und nach einem Programmstopp über Neustartdaten überwacht werden soll.

[\GO3]

Group Output 3

Datentyp: signalgo

Die Signalvariable für ein digitales Gruppenprozesssignal, die auf null gesetzt und nach einem Programmstopp über Neustartdaten überwacht werden soll.

[\GO4]

Group Output 4

Datentyp: signalgo

Die Signalvariable für ein digitales Gruppenprozesssignal, die auf null gesetzt und nach einem Programmstopp über Neustartdaten überwacht werden soll.

Mindestens einer der optionalen Parameter D01, GO1 ... GO4 muss verwendet werden.

ShadowDO

Shadow Digital Output

Datentyp: signaldo

Die Signalvariable für ein digitales Signal, die angibt, ob der Prozess auf der Bahn aktiv ist.

Das Signal wird bei einem STOP oder QSTOP nicht von dem Prozess TriggStopProc auf null gesetzt, sondern seine Werte werden in restartdata wiedergegeben.

Programmabarbeitung

Einstellung und Abarbeitung von TriggStopProc

TriggStopProc muss von den folgenden zwei Routinen aufgerufen werden:

- der START-Ereignisroutine oder im Initialisierungsteil des Programms (PP auf „main“ setzen und den internen Prozess für TriggStopProc beenden)
- der POWERON-Ereignisroutine (durch das Setzen von Power Off wird der interne Prozess für TriggStopProc unterbrochen)

Der interne Name des Prozesses für TriggStopProc entspricht dem des Signalnamens im Argument ShadowDO. Wenn TriggStopProc mit dem gleichen Signalnamen im Argument ShadowDO in derselben oder einer anderen Programmtask zweimal abgearbeitet wird, ist nur das letzte abgearbeitete TriggStopProc aktiv.

Die Abarbeitung von TriggStopProc startet die Überwachung von E/A-Signalen nur bei STOP und QSTOP.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.312 TriggStopProc - Generieren von Neustartdaten für Trigg-Signale bei einem Stopp
RobotWare - OS
Fortsetzung

Programmstopp STOP

Der Prozess TriggStopProc besteht aus folgenden Schritten:

1. Warten auf den Stillstand des Roboters auf der Bahn
2. Speichern des aktuellen Werts (Vorwert laut restartdata) aller verwendeten Prozesssignale. Setzen aller verwendeten Prozesssignale außer ShadowDO. auf null.
3. Im nächsten Zeitschlitz, ca. 500 ms, Folgendes erledigen, wenn einige Prozesssignale ihren Wert in dieser Zeit ändern:
   - Den aktuellen Wert erneut speichern (Nachwert laut restatdata)
   - Dieses Signal auf null setzen, es sei denn, es handelt sich um ShadowDO
   - Anzahl der Wertübergänge (Flanken) des Signals ShadowDO zählen
4. Die angegebene persistente Variable mit Neustartdaten aktualisieren.

Not-Halt (QSTOP)

Der Prozess TriggStopProc besteht aus folgenden Schritten:

1. Den nächsten Schritt so bald wie möglich ausführen.
2. Speichern des aktuellen Werts (Vorwert laut restartdata) aller verwendeten Prozesssignale. Setzen aller verwendeten Prozesssignale außer ShadowDO. auf null
3. Im nächsten Zeitschlitz, ca. 500 ms, Folgendes erledigen, wenn einige Prozesssignale ihren Wert in dieser Zeit ändern:
   - Den aktuellen Wert erneut speichern (Nachwert laut restatdata
   - Dieses Signal auf null setzen, es sei denn, es handelt sich um ShadowDO
   - Anzahl der Wertübergänge (Flanken) des Signals ShadowDO zählen
4. Die angegebene persistente Variable mit Neustartdaten aktualisieren.

Kritischer Bereich für Prozessneustart

Sowohl im Roboterservo als auch in den externen Geräten sind Verzögerungen vorhanden. Alle Instruktionen in der Trigg-Familie sind so konzipiert, dass alle Signale an der richtigen Stelle in der Roboterbahn unabhängig von unterschiedlichen Verzögerungen in externen Geräten gesetzt werden, um bestmögliche Prozessergebnisse zu erzielen. Daher kann das Setzen von E/A-Signalen nach dem Stillstand des Roboters bei Programmstopp (STOP) oder nach dem Registrieren eines Not-Halt (QSTOP) systemintern um 0 – 80 ms verzögert werden. Wegen dieses Nachteils für die Neustartfunktion werden der Vorwert, der Nachwert und auch die Schattenflanken in die Neustartdaten einbezogen.

Wenn dieser kritische Zeitschlitz von 0 - 80 ms mit den folgenden Anwendungsprozessfällen zusammenfällt, kann ein guter Prozessneustart nur schwer durchgeführt werden:

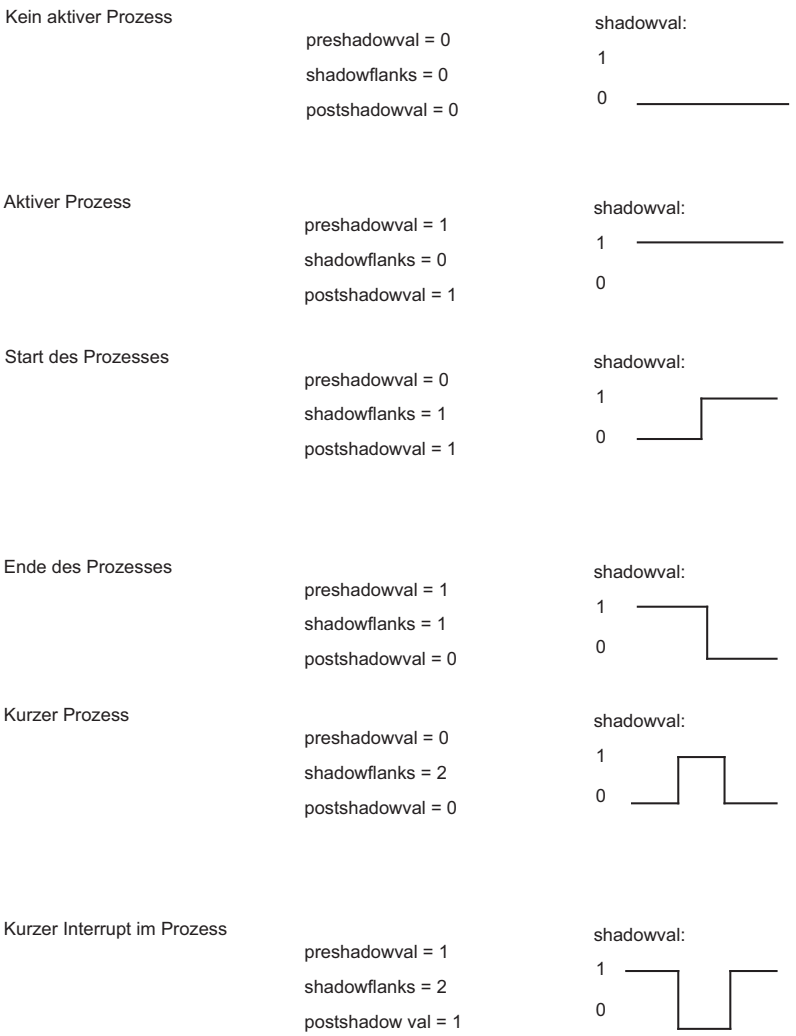
- Anfang des Anwendungsprozesses
- Ende des Anwendungsprozesses
- innerhalb eines kurzen Anwendungsprozesses

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- während eines kurzen Interrupts im Anwendungsprozess

Die folgende Abbildung veranschaulicht die Prozessphasen bei STOP oder QSTOP im kritischen Zeitfenster von 0–80 ms.



xx0500002326

Ausführen eines Neustarts

Der Neustart von Prozessinstruktionen (NOSTEPIN-Routinen) auf der Roboterbahn muss in einer RESTART-Ereignisroutine durchgeführt werden.

Die RESTART-Ereignisroutine kann aus folgenden Schritten bestehen:

	Aktion
1.	Nach einem QSTOP wird die Rückkehr auf die Bahn am Anfang des Programms durchgeführt.
2.	Analyse der Neustartdaten aus dem letzten STOP oder QSTOP.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.312 TriggStopProc - Generieren von Neustartdaten für Trigg-Signale bei einem Stopp
RobotWare - OS
Fortsetzung

	Aktion
3.	Festlegung der Strategie für den Prozessneustart durch die Ergebnisse der Analyse, wie z. B.: • Prozess aktiv, Prozessneustart durchführen • Prozess nicht aktiv, Prozessneustart nicht durchführen • Je nach Typ der Prozessanwendung angemessene Maßnahmen ergreifen: - Start des Prozesses - Ende des Prozesses - Kurzer Prozess - Kurze Unterbrechung des Prozesses
4.	Rückwärtsschritte auf der Bahn.
5.	Beibehalten der Programmergebnisse beim Neustart der Bewegung.

Beim Warten in einer `STOP`-oder `QSTOP`-Ereignisroutine, bis der `TriggStopProc`-Prozess z. B. mit `WaitUntil (myproc.restartstop=TRUE), \MaxTime:=2;` bereit ist, muss der Benutzer immer das Flag in der `RESTART`-Ereignisroutine mit z. B. `myproc.restartstop:=FALSE` zurücksetzen. Anschließend kann der Neustart durchgeführt werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Einschränkungen

Nach einem Stromausfall wird der Neustart von Prozessinstruktionen nicht unterstützt.

`TriggStopProc` kann nicht in einer UNDO-Behandlung oder RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart, Reset oder Step.

Syntax

```
TriggStopProc
  [ RestartRef ':=' ] < persistent (PERS) of restartdata>
  [ '\' DO1 ':=' < variable (VAR) of signaldo>
  [ '\' GO1 ':=' < variable (VAR) of signalgo> ]
  [ '\' GO2 ':=' < variable (VAR) of signalgo> ]
  [ '\' GO3 ':=' < variable (VAR) of signalgo> ]
  [ '\' GO4 ':=' < variable (VAR) of signalgo> ] ','
  [ ShadowDO ':=' ] < variable (VAR) of signaldo> ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Prozessinstruktionen	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Neustartdaten	*restartdata - Neustartdaten für Trigg-Signale auf Seite 1768*
Rückwärtsschritte auf der Bahn	*StepBwdPath - Rückwärtsbewegung um einen Schritt auf der Bahn auf Seite 821*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.313 TryInt - Überprüfen, ob ein Datenobjekt eine gültige Ganzzahl ist

Verwendung

`TryInt` wird verwendet, um zu überprüfen, ob ein Datenobjekt eine gültige Ganzzahl ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `TryInt`:

Beispiel 1:

```
VAR num myint := 4;
...
TryInt myint;
```

Der Wert von `myint` wird überprüft, und weil `4` eine gültige Ganzzahl ist, wird die Programmabarbeitung fortgesetzt.

Beispiel 2:

```
VAR dnum mydnum := 20000000;
...
TryInt mydnum;
```

Der Wert von `mydnum` wird überprüft, und weil `20000000` eine gültige dnum-Ganzzahl ist, wird die Programmabarbeitung fortgesetzt.

Beispiel 3

```
VAR num myint := 5.2;
...
TryInt myint;
...
ERROR
  IF ERRNO = ERR_INT_NOTVAL THEN
    myint := Round(myint);
    RETRY;
  ENDIF
```

Der Wert von `myint` wird überprüft, und weil `5.2` keine gültige Ganzzahl ist, wird ein Fehler ausgelöst. In der Fehlerbehandlung wird `myint` auf 5 gerundet und die Instruktion `TryInt` wird erneut ausgeführt.

Argumente

```
TryInt DataObj | DataObj2
```

`DataObj`

Data Object

Datentyp: `num`

Das Datenobjekt, für das überprüft werden soll, ob es eine gültige Ganzzahl ist.

`DataObj2`

Data Object 2

Datentyp: `dnum`

Das Datenobjekt, für das überprüft werden soll, ob es eine gültige Ganzzahl ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Das angegebenen Datenobjekt wird überprüft:

- Wenn es eine gültige Ganzzahl ist, fährt die Abarbeitung mit der nächsten Instruktion fort.
- Wenn es eine keine gültige Ganzzahl ist, fährt die Abarbeitung mit der Fehlerbehandlungsroutine in der gegenwärtigen Prozedur fort.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_INT_NOTVAL	DataObj enthält einen Dezimalwert.
ERR_INT_MAXVAL	• Der Wert von DataObj ist größer oder kleiner als der Ganzzahl-Wertebereich des Datentyps num. • Der Wert von DataObj2 ist größer oder kleiner als der Ganzzahl-Wertebereich des Datentyps dnum.

Beachten Sie, dass der Wert 3.0 als Ganzzahl ausgewertet wird, weil der Teil .0 ignoriert werden kann.

Syntax

```
TryInt
  [ DataObj ':=' ] < expression (IN) of num>
  | [ DataObj2 ':=' ] < expression (IN) of dnum>' ;'
```

Weitere Informationen

Information über	Siehe
Datentyp num	*num - Numerische Werte auf Seite 1740*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.314 TRYNEXT - Überspringen einer fehlerverursachenden Instruktion

Verwendung

Die Instruktion The TRYNEXT wird verwendet, um die Abarbeitung nach einem Fehler wieder aufzunehmen, beginnend mit der Instruktion nach der fehlerverursachenden Instruktion.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TryNext`:

Beispiel 1:

```
reg2 := reg3/reg4;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    reg2:=0;
    TRYNEXT;
  ENDIF
```

Es wird versucht, `reg3` durch `reg4` zu dividieren. Wenn `reg4` gleich `0` ist (Division durch null), erfolgt ein Sprung zur Fehlerbehandlungsroutine, wobei `reg20` zugewiesen wird. Die Instruktion `TRYNEXT` wird dann verwendet, um mit der nächsten Instruktion fortzufahren.

Programmabarbeitung

Die Programmabarbeitung fährt mit der Instruktion fort, die auf die fehlerverursachende Instruktion folgt.

Einschränkungen

Die Instruktion ist nur in der Fehlerbehandlung innerhalb einer Routine möglich.

Syntax

```
TRYNEXT';'
```

Weitere Informationen

Information über	Siehe
Fehlerbehandlungen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.315 TuneReset - Zurücksetzen der Servo-Abstimmung

Verwendung

`TuneReset` wird verwendet, um das dynamische Verhalten aller Roboterachsen und ihrer externen mechanischen Einheiten auf ihre normalen Werte zurückzusetzen.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TuneReset`:

Beispiel 1:

```
TuneReset;
```

Zurücksetzen der Abstimmungswerte auf 100 % für alle Achsen.

Programmabarbeitung

Die Abstimmungswerte werden für alle Achsen auf 100 % zurückgesetzt.

Die Standard-Servo-Abstimmungswerte für alle Achsen werden automatisch mit der Instruktion `TuneReset` gesetzt.

- bei einem **Neustart**.
- beim Laden eines neuen Programms
- und bei Beginn der Programmabarbeitung vom Anfang.

Syntax

```
TuneReset ';'
```

Weitere Informationen

Information über	Siehe
Abstimmen von Servos	*TuneServo - Abstimmen von Servos auf Seite 1004*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.316 TuneServo - Abstimmen von Servos

Verwendung

`TuneServo` wird verwendet, um die Dynamik separater Achsen am Roboter einzustellen.

Für die meisten Anwendungen ist die Benutzung von `TuneServo` nicht notwendig, allerdings wird bei einigen Anwendungen `TuneServo` zum Erreichen der gewünschten Genauigkeit benötigt. Die Benutzung von `TuneServo` kann in vielen Fällen durch die Auswahl eines vordefinierten *Motion Process Mode* ersetzt werden, siehe *Technisches Referenzhandbuch - Systemparameter*, oder durch die Modifizierung eines vordefinierten *Motion Process Mode*.

Bei externen Achsen kann `TuneServo` zur Lastanpassung verwendet werden.

Vermeiden Sie `TuneServo`-Befehle, während sich der Roboter bewegt. Dies kann vorübergehend zu hohen Drehmomenten und damit zu Fehlern und Stopps führen.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Hinweis

Für optimale Abstimmung ist es wesentlich, dass die korrekten Lastdaten verwendet werden. Prüfen Sie dies, bevor Sie `TuneServo` verwenden.

WARNUNG

Falsche Anwendung von `TuneServo` kann oszillierende Bewegungen oder Drehmomente verursachen, die den Roboter beschädigen können. Bedenken Sie dies stets und gehen Sie bei der Verwendung von `TuneServo` sehr umsichtig vor.

Beschreibung

Verringert Überschwingen und Vibrationen – `TUNE_DF`

`TUNE_DF` kann verwendet werden, um die erwartete mechanische Resonanzfrequenz einer bestimmten Achse anzupassen. Ein Abstimmungswert von 95 % verringert die Resonanzfrequenz um 5 %. `TUNE_DF` wird meistens eingesetzt, um die unzureichende Festigkeit eines Fundaments auszugleichen, z.B. bei einem flexiblen Fundament. In diesem Fall wird der Abstimmungswert der Achsen 1 und 2 reduziert, normalerweise auf einen Wert zwischen 80 % und 99 %.

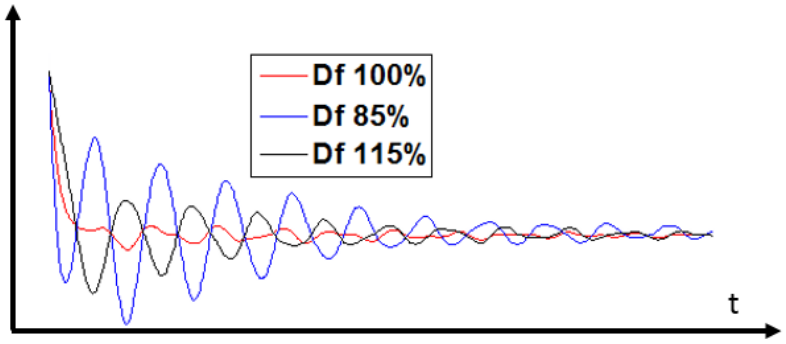
Die Verwendung von `TUNE_DF` für die Achsen 3 - 6 ist selten und wird normalerweise nicht empfohlen. Eine Ausnahme bildet das Einstellen der Achsen 4 - 6, wenn die Resonanzfrequenz einer erhöhten flexiblen Nutzlast kompensiert werden soll.

Bei einer korrekten Anpassung, die nicht zu hoch und nicht zu niedrig ist, verringert `TUNE_DF` Überschwingungen und Vibrationen. Beim Anpassen von `TUNE_DF` vorsichtig sein, da ein zu hoher oder zu niedriger Wert die Bewegung deutlich

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

beeinträchtigen kann. Ein Beispiel findet sich in der Abbildung unten. In diesem Fall erreicht ein Abstimmungswert von 100 % die besten Ergebnisse.



xx1400001280

Der Abstimmungswert kann durch die Verwendung von TuneMaster automatisch optimiert werden, dies wird empfohlen.

Folgender RAPID-Codeausschnitt zur Einstellung von Achse 1 ist ein Beispiel für manuelles Abstimmen:

```
MoveAbsJ [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
     v200, fine, myTool;
FOR DF FROM 80 TO 100 STEP 5 DO
  TuneServo ROB_1,1,DF\Type:=TUNE_DF;
  MoveAbsJ [[2,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
       vmax, fine, myTool;
  WaitTime 1;
  MoveAbsJ [[0,0,0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],
       vmax, fine, myTool;
  WaitTime 1;
ENDFOR
TuneReset;
```

Hier wurde der Abstimmungswert in 5 %-Schritten verändert, auch 2 %-Schritte sind möglich. Beachten Sie, dass es eine kleine Bewegung sein sollte, 2 Grad ist ein typischer Wert. Positionieren Sie den Roboter in einer typischen Arbeitsbereichsposition. Wählen Sie den Abstimmungswert, der bei einer Sichtprüfung Überschwingungen und Vibrationen minimiert.

Auch durch eine Senkung des Abstimmungswerts für `TUNE_DH` oder durch eine Senkung der Beschleunigung durch `AccSet` können Überschwingungen und Vibrationen reduziert werden. Dies ist in vielen Fällen die beste Lösung. Wenn ein Problem jedoch nur mit `TUNE_DF` gelöst werden kann, wird die Zykluszeit nicht beeinflusst und die Verwendung von `TUNE_DF` ist dann die beste Lösung.

Bei Robotern mit *Mounting Stiffness Factor*-Möglichkeit, siehe *Motion Process Mode* in *Technisches Referenzhandbuch - Systemparameter*, wird die Verwendung von *Mounting Stiffness Factor* durch die Benutzung von `TUNE_DF` zur Kompensierung flexibler Fundamente ersetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

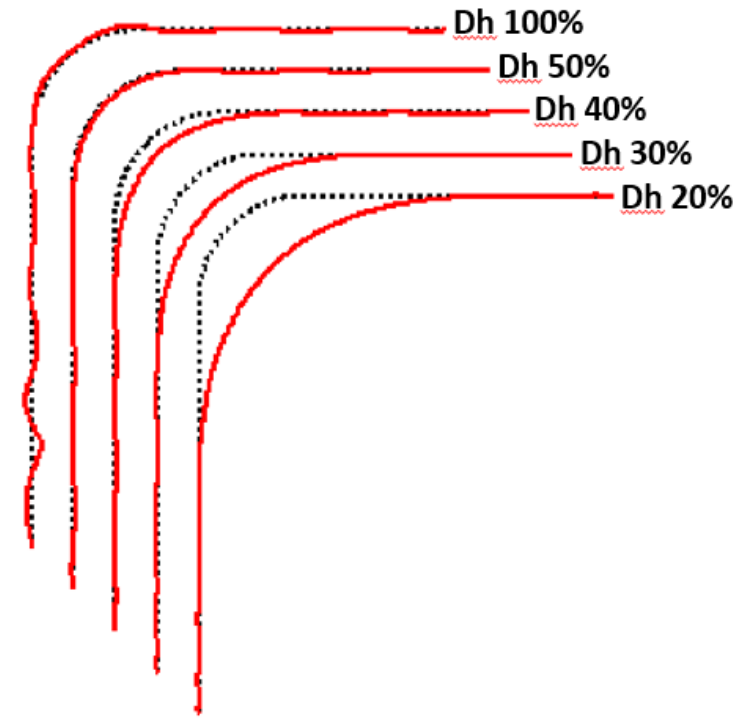
1.316 TuneServo - Abstimmen von Servos
RobotWare - OS
Fortsetzung

Verringert Überschwingen und Vibrationen – TUNE_DH

Mithilfe von TUNE_DH kann die Glätte der Roboterbahn durch Anpassen der effektiven Bandbreite des Systems erhöht werden. Der Abstimmungswert kann nur verringert werden und Werte über 100 % haben keinen Einfluss auf die Bewegungen. Ein Abstimmungswert unter 100 % vermindert die Bandbreite und erhöht die Glätte, wodurch Überschwingungen und Vibrationen verringert werden.

TUNE_DH erhöht die Zykluszeit nur in Feinpunkten, während die Beschleunigung die Zykluszeit entlang der ganzen Roboterbahn erhöht. Daher kann die Benutzung von TUNE_DH ein sehr zykluszeiteffizienter Weg für die Verringerung von Vibrationen und Überschwingungen sein, im Gegensatz zu der Verringerung der Beschleunigung durch die Verwendung der AccSet-Instruktion. Bei hoher Geschwindigkeit werden bei der Verwendung von TUNE_DH höhere Verschleifzonen als programmiert erkennbar. Folglich reduziert die Verwendung von TUNE_DH durch Vibrationen verursachte Bahnfehler, erhöht jedoch durch Abkürzungen in Verschleifzonen die Anzahl der Bahnfehler bei hoher Geschwindigkeit. Die Grösse der Abkürzungen erhöht sich mit abnehmendem Abstimmungswert und zunehmender Geschwindigkeit. Wenn diese Abkürzungen nicht zulässig sind, wird AccSet anstelle von TUNE_DH empfohlen.

Die Abbildung unten zeigt die Auswirkung eines verringerten Abstimmungswertes und dass eine unerwünschte Vibration durch den korrekten Abstimmungswert beseitigt werden kann. Bei geringeren Abstimmungswerten wird der Tastenbefehl in der Verschleifzone sichtbar.



xx1400001281

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Es genügt, die Instruktion `TuneServo` mit dem Argument `\Type:=TUNE_DH` für eine Achse auszuführen. Alle Achsen in derselben mechanischen Einheit erhalten automatisch denselben Abstimmungswert.

Beispiele:

- Schneiden mit TCP-Geschwindigkeiten von bis zu 300 mm/s. Ein Abstimmungswert von 50 % verringert unerwünschte Vibrationen.

  Dies wird manchmal kombiniert mit `AccSet`, z. B. `AccSet 50,100;`.
- Materialhandhabung bei hoher Geschwindigkeit. Ein Abstimmungswert von 15 % reduziert unerwünschte Vibrationen.

VORSICHT

Ändern Sie niemals den Abstimmungswert während sich der Roboter bewegt, und seien Sie bei der Verwendung von geringen Abstimmungswerten (weniger als 30 %) vorsichtig, da der Roboter in Verschleifzonen Tastenbefehle verwendet.

Nur für ABB-interne Verwendung – `TUNE_DK, TUNE_DL, TUNE_DG, TUNE_DI`

WARNUNG

Nur für ABB-interne Verwendung. Verwenden Sie diese Abstimmungstypen nicht. Falsche Anwendung kann oszillierende Bewegungen oder Drehmomente verursachen, die den Roboter beschädigen können.

Abstimmen der externen Achsen - `TUNE_KP`, `TUNE_KV`, `TUNE_TI`

Diese Abstimmungstypen beeinflussen die Positionskontrolle (kp), die Geschwindigkeitskontrolle (kv) und die Integrationszeit (ti) für externe Achsen. Diese Werte dienen der Anpassung externer Achsen an verschiedene Lastenträgheitsmomente. Die grundlegende Abstimmung von externen Achsen kann durch diese Abstimmungstypen vereinfacht werden.

Abstimmen der Roboterachsen – `TUNE_KP, TUNE_KV, TUNE_TI`

Diese Parameter können verwendet werden, um das Verhalten der Servo-Steuerung zu ändern. `TUNE_KP` wirkt sich auf die äquivalente Verstärkung der Positionssteuerung aus, `TUNE_KV` wirkt sich auf die äquivalente Verstärkung der Geschwindigkeitssteuerung aus, und `TUNE_TI` wirkt sich auf das integrale Verhalten der Steuerung aus.

Durch das Erhöhen des Abstimmungswertes für `TUNE_KV` wird die Servo-Steifigkeit des Roboters erhöht. Dies kann bei Kontaktanwendungen von Nutzen sein, da die vollständige Steifigkeit des Robotersystems sowohl von der Servo-Steifigkeit also auch von der mechanischen Steifigkeit abhängt. Ein erhöhter Abstimmungswert für `TUNE_KV` verringert zudem Bahnfehler und kann bei Schneide- und Schweißanwendungen von Nutzen sein, bei denen die Geschwindigkeit weniger als 100 mm/s beträgt. Typische Abstimmungswerte sind 150% - 200%. Ein zu hoher Abstimmungswert verursacht Motorvibrationen und muss vermieden werden. Gehen Sie beim Einstellen von `TUNE_KV` stets vorsichtig vor und achten Sie dabei auf erhöhte Motorgeräusche. Verwenden Sie zudem keine höheren Abstimmungswerte als für die Erfüllung der Anwendungsanforderungen benötigt

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

werden. Ein zu hoher Abstimmungswert kann zudem zu erhöhten Vibrationen aufgrund von mechanischen Resonanzen führen.

Ein erhöhter Abstimmungswert für TUNE_KP und ein verringerter Abstimmungswert für TUNE_TI erhöht die Servo-Steifigkeit und verringert Bahnfehler bei niedriger Geschwindigkeit im Niederfrequenzbereich. Typische Abstimmungswerte für TUNE_KP sind 150 % bis 300 % und für TUNE_TI 20 % bis 50 %. In den meisten Fällen ist TUNE_KV der wichtigste Parameter und TUNE_KP und TUNE_TI erfordern keine Anpassung. Ein zu hoher Abstimmungswert für TUNE_KP oder ein zu niedriger Abstimmungswert für TUNE_TI kann zudem zu erhöhten Vibrationen aufgrund von mechanischen Resonanzen führen.

Beispiel:

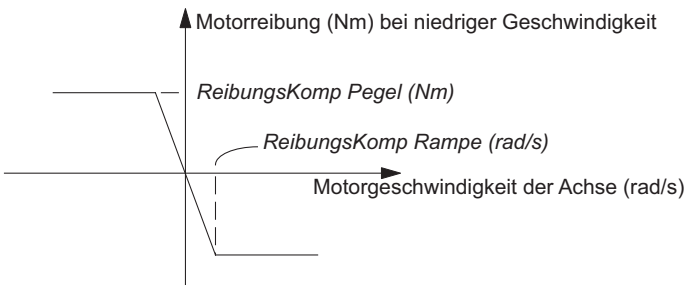
- Bei Entgratanwendungen ist eine höhere Servo-Steifigkeit des Roboters erforderlich, um Bahnfehler zu verringert. TUNE_KV 175 %, TUNE_KP 250 % und TUNE_TI 30 %.

  Dies wird häufig kombiniert mit AccSet, z. B. `AccSet 30,100;`.

Reibungskompensierung - TUNE_FRIC_LEV, TUNE_FRIC_RAMP

Diese Abstimmungstypen können Sie zur Verringerung von Bahnfehlern verwenden, die durch Reibung und Spiel bei niedrigen Geschwindigkeiten (10 bis 200 mm/s) verursacht werden. Diese Bahnfehler treten auf, wenn eine Roboterachse die Bewegungsrichtung ändert. Aktivieren Sie die Reibungskompensierung für eine Achse, indem Sie den Systemparameter Motion/Control Parameters/Friction FFW On auf Yes setzen.

Das Reibungsmodell ist eine konstante Größe mit entgegengesetztem Vorzeichen der Richtung der Achsengeschwindigkeit. *Friction FFW Level (Nm)* ist die absolute Reibungsgröße bei (niedrigen) Geschwindigkeiten und ist größer als *Friction FFW Ramp (rad/s)*. Siehe Abbildung unten, die ein Reibungsmodell zeigt.



xx0500002188

TUNE_FRIC_LEV überschreibt den Wert des Systemparameters *Friction FFW Level*.

Durch Abstimmen von *Friction FFW Level* (mithilfe von TUNE_FRIC_LEV) für jede Roboterachse kann die Bahngenauigkeit des Roboters im Geschwindigkeitsbereich zwischen 20 und 100 mm/s deutlich verbessert werden. Für größere Roboter (insbesondere die Serie IRB 6400) ist die Wirkung jedoch minimal, da andere Quellen von Tracking-Fehlern bei diesen Robotern vorherrschen.

TUNE_FRIC_RAMP überschreibt den Wert des Systemparameters *Friction FFW Ramp*. In den meisten Fällen braucht *Friction FFW Ramp* nicht eingestellt zu werden. Die Standardeinstellung ist angemessen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Stimmen Sie nur jeweils eine Achse ab. Ändern Sie den Abstimmungswert in kleinen Schritten, um die Ebene zu ermitteln, die eine Minimierung der Bahnabweichung des Roboters an den Positionen auf der Bahn ergibt, wo die entsprechende Achse die Bewegungsrichtung ändert. Wiederholen Sie diese Prozedur für die nächste Achse usw.

Die endgültigen Abstimmungswerte können in die Systemparameter übertragen werden. Beispiel:

- Friction FFW Level = 1. Endgültiger Abstimmungswert (`TUNE_FRIC_LEV`) = 150 %.
- Setzen Sie Friction FFW Level = 1,5 und Abstimmungswert = 100 % (Standard), was entsprechend ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `TuneServo`:

Beispiel 1:

```
TuneServo MHA160R1, 1, 110 \Type:= TUNE_KP;
```

Aktivieren des Abstimmungstyps `TUNE_KP` mit dem Abstimmungswert `110` % auf Achse `1` in der mechanischen Einheit `MHA160R1`.

Argumente

```
TuneServo MecUnit Axis TuneValue [\Type]
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

`TuneValue`

Datentyp: `num`

Abstimmungswert in Prozent (1-500). Der normale Wert ist 100 %.

`[ \Type ]`

Datentyp: `tunetype`

Typ der Servo-Abstimmung. Verfügbare Typen sind `TUNE_DF`, `TUNE_KP`, `TUNE_KV`, `TUNE_TI`, `TUNE_FRIC_LEV`, `TUNE_FRIC_RAMP`, `TUNE_DG`, `TUNE_DH`, `TUNE_DI`. Typ `TUNE_DK` und `TUNE_DL` nur für ABB-interne Verwendung.

Dieses Argument kann nur bei Verwendung des Abstimmungstyps `TUNE_DF` weggelassen werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.316 TuneServo - Abstimmen von Servos

RobotWare - OS

Fortsetzung

Programmabarbeitung

Angegebener Abstimmungstyp und Abstimmungswert werden für die angegebene Achse aktiviert. Dieser Wert gilt für alle Bewegungen, bis ein neuer Wert für die aktuelle Achse programmiert wird oder bis die Abstimmungstypen und -werte für alle Achsen mithilfe der Instruktion `TuneReset` zurückgesetzt werden.

Die Standard-Servo-Abstimmungswerte für alle Achsen werden automatisch mit der Instruktion `TuneReset` gesetzt.

- bei einem **Neustart**.
- beim Laden eines neuen Programms
- und bei Beginn der Programmabarbeitung vom Anfang.

Einschränkungen

Bei einem Stromausfall wird sämtliche aktive Servo-Abstimmung auf die Standardwerte gesetzt.

Diese Einschränkung kann im Anwenderprogramm beim Neustart nach einem Stromausfall behandelt werden.

Syntax

```
TuneServo
  [MecUnit ':=' ] < variable (VAR) of mecunit> ','
  [Axis ':=' ] < expression (IN) of num> ','
  [TuneValue ':=' ] < expression (IN) of num>
  ['\' Type ':=' <expression (IN) of tunetype>] ';'
```

Weitere Informationen

Information über	Siehe
Andere Bewegungseinstellungen	*Technisches Referenzhandbuch - RAPID Overview*
Typen der Servo-Abstimmung	*tunetype – Servo-Abstimmungstyp auf Seite 1848*
Zurücksetzen aller Servo-Abstimmungen	*TuneReset - Zurücksetzen der Servo-Abstimmung auf Seite 1003*
MotionProcessModeSet – Einstellen des Bewegungsprozess-Modus.	*MotionProcessModeSet - Einstellen des Bewegungsprozess-Modus auf Seite 377*
Einstellen der externen Achsen	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*
Reibungskompensierung	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.317 UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ

Verwendung

`UIMsgBox` (*User Interaction Message Box*) wird verwendet, um mit dem Benutzer des Robotersystems über ein verfügbares Benutzergerät wie das FlexPendant zu kommunizieren. An den Bediener wird eine Meldung ausgegeben, der durch Drücken einer Taste antwortet. Die Benutzerauswahl wird dann zurück an das Programm übertragen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `UIMsgBox`:

Siehe auch *Weitere Beispiele auf Seite 1017*.

Beispiel 1:

```
UIMsgBox "Continue the program ?";
```

Die Meldung `"Continue the program ?"` wird angezeigt. Das Programm fährt fort, wenn der Benutzer die Standardtaste OK drückt.

Beispiel 2:

```
VAR btnres answer;
...
UIMsgBox
  \Header:="UIMsgBox Header",
  "Message Line 1"
  \MsgLine2:="Message Line 2"
  \MsgLine3:="Message Line 3"
  \MsgLine4:="Message Line 4"
  \MsgLine5:="Message Line 5"
  \Buttons:=btnOKCancel
  \Icon:=iconInfo
  \Result:=answer;
IF answer = resOK my_proc;
```

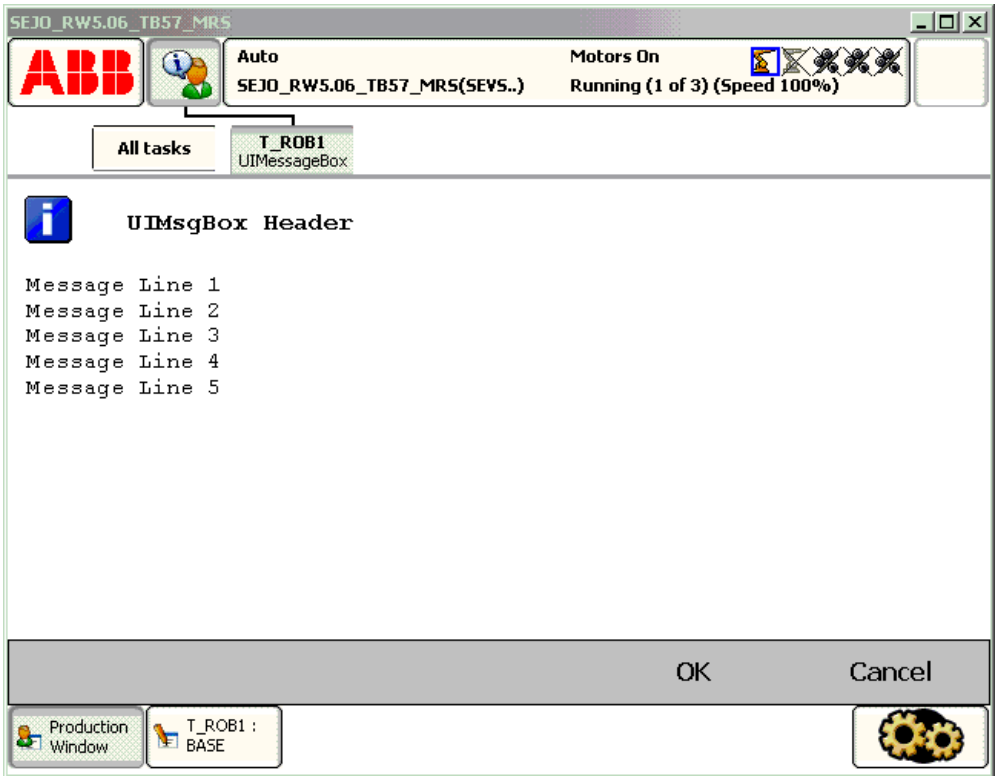
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.317 UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ

RobotWare - OS

Fortsetzung



xx0500002432

Obiges Meldungsfenster wird auf dem FlexPendant mit dem Icon, der Überschrift, den Meldungszeilen 1 bis 5 und den Drucktasten angezeigt. Die Programmabarbeitung wartet, bis „OK“ oder „Abbrechen“ gedrückt wird. Anders ausgedrückt: `answer` wird 1 (OK) oder 5 (Abbrechen) zugewiesen, je nachdem, welche Taste gedrückt wird. Wenn die Antwort (answer) OK lautet, wird `my_proc` aufgerufen.

Beachten Sie, dass die Meldungszeilen 1 bis 5 in separaten Zeilen angezeigt werden (`\Wrap` wird nicht verwendet).

Argumente

UIMsgBox [\Header] MsgLine1 [\MsgLine2] [\MsgLine3] [\MsgLine4] [\MsgLine5] [\Wrap] [\Buttons] [\Icon] [\Image] [\Result] [\MaxTime] [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive] [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag] [\UIActiveSignal]

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`MsgLine1`

Message Line 1

Datentyp: `string`

Textzeile 1, die auf dem Display angezeigt wird. Max. 55 Zeichen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\MsgLine2]`

Message Line 2

Datentyp: `string`

Eine zusätzliche zweite Textzeile, die am Display angezeigt wird. Max. 55 Zeichen.

`[\MsgLine3]`

Message Line 3

Datentyp: `string`

Eine zusätzliche dritte Textzeile, die am Display angezeigt wird. Max. 55 Zeichen.

`[\MsgLine4]`

Message Line 4

Datentyp: `string`

Eine zusätzliche vierte Textzeile, die am Display angezeigt wird. Max. 55 Zeichen.

`[\MsgLine5]`

Message Line 5

Datentyp: `string`

Eine zusätzliche fünfte Textzeile, die am Display angezeigt wird. Max. 55 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Bei Auswahl werden die Zeichenfolgen `MsgLine1 ... MsgLine5` zu einer Zeichenfolge zusammengefasst und auf so wenig Zeilen wie möglich verteilt, wobei die einzelnen Zeichenfolgen nur durch ein einzelnes Leerzeichen voneinander getrennt werden.

Standardmäßig wird jede Meldungszeichenfolge *MsgLine1 ... MsgLine5* auf dem Display in einer separaten Zeile angezeigt.

`[\Buttons]`

Datentyp: `buttondata`

Gibt die anzuzeigenden Schaltflächen an. Es kann nur eine der vordefinierten Schaltflächenkombinationen des Typs `buttondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1016*.

Standardmäßig zeigt das System die Taste OK an. (`\Buttons:=btn OK`).

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1016*.

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.317 UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ
RobotWare - OS
Fortsetzung

Der Name des zu verwendenden Bildes. Um eigene Bilder zu starten, müssen die Bilder im Verzeichnis `HOME:` des aktiven Systems oder direkt im aktiven System abgelegt werden.

Es wird empfohlen, die Dateien im Verzeichnis `HOME:` abzulegen, so dass sie bei einem Backup and Restore gespeichert werden.

Ein **Neustart** muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

`[\Result]`

Datentyp: `btnres`

Die Variable, für die abhängig von der gedrückten Taste der numerische Wert 0…7 zurückgegeben wird. Nur eine der vordefinierten Konstanten vom Typ `btnres` kann verwendet werden, um die Benutzerauswahl zu testen. Siehe *Vordefinierte Daten auf Seite 1016*.

Bei einer Systemunterbrechung wie `\MaxTime`, `\DIBreak` oder `\DOBreak`, oder wenn `\Buttons:=btnNone, resUnkwn` mit dem Wert 0 zurückgegeben wird.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn innerhalb dieser Zeitdauer keine Taste gewählt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOBreak]`

Digital Output Break

Datentyp: `signaldo`

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOPassive]`

Digital Output Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DOBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DOBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolBreak]`

Persistent Boolean Break

Datentyp: `bool`

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolPassive]`

Persistent Boolean Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `PersBoolBreak` verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn die persistente boolesche Variable `PersBoolBreak` auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.317 UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ
RobotWare - OS
Fortsetzung

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignal verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Das Meldungsfenster mit dem Icon, dem Header, den Meldungszeilen, Bild und Tasten wird gemäß den programmierten Argumenten angezeigt. Die Programmabarbeitung wartet, bis der Benutzer eine Taste betätigt oder das Meldungsfenster durch einen Timeout oder ein Signal unterbrochen wird. Die Auswahl des Benutzers und der Grund der Unterbrechung werden in das Programm zurückübertragen.

Neues Meldungsfenster auf Interrupt-Ebene übernimmt den Fokus vom Meldungsfenster der Basisebene.

Vordefinierte Daten

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;

!Buttons:
  CONST buttondata btnNone := -1;
  CONST buttondata btnOK := 0;
  CONST buttondata btnAbrtRtryIgn := 1;
  CONST buttondata btnOKCancel := 2;
  CONST buttondata btnRetryCancel := 3;
  CONST buttondata btnYesNo := 4;
  CONST buttondata btnYesNoCancel := 5;

!Results:
  CONST btnres resUnkwn := 0;
  CONST btnres resOK := 1;
  CONST btnres resAbort := 2;
  CONST btnres resRetry := 3;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
CONST btnres resIgnore := 4;
CONST btnres resCancel := 5;
CONST btnres resYes := 6;
CONST btnres resNo := 7;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen in der Fehlerbehandlungsroutine behandelt werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion UIMsgBox sind unten aufgeführt.

Beispiel 1:

```
VAR errnum err_var;
...
UIMsgBox \Header:= "Example 1", "Waiting for a break condition..."
     \Buttons:=btnNone \Icon:=iconInfo \MaxTime:=60 \DIBreak:=di5
     \BreakFlag:=err_var;

TEST err_var
CASE ERR_TP_MAXTIME:
  ! Time out break, max time 60 seconds has elapsed
CASE ERR_TP_DIBREAK:
  ! Input signal break, signal di5 has been set to 1
DEFAULT:
  ! Not such case defined
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.317 UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ

RobotWare - OS

Fortsetzung

```
ENDTEST
```

Die Meldungsbox wird angezeigt, bis eine Pausenvoraussetzung erfüllt ist. Der Bediener kann die Meldungsbox nicht beantworten oder entfernen, da `btnNone` für das Argument `\Buttons` festgesetzt ist. Die Meldungsbox wird entfernt, sobald `di5` für 1 oder Timeout festgesetzt ist (nach `60`Sekunden).

Beispiel 2:

```
VAR errnum err_var;
...
UIMsgBox \Header:= "Example 2", "Waiting for a break condition..."
     \Buttons:=btnNone \Icon:=iconInfo \MaxTime:=60 \DIBreak:=di5
     \DIPassive \BreakFlag:=err_var;

TEST err_var
CASE ERR_TP_MAXTIME:
  ! Time out break, max time 60 seconds has elapsed
CASE ERR_TP_DIBREAK:
  ! Input signal break, signal di5 has been set to 0
DEFAULT:
  ! Not such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt, bis eine Unterbrechungsbedingung true ist. Der Bediener kann nicht auf das Meldungsfenster reagieren oder es entfernen, da `btnNone` für das Argument `\Buttons` gesetzt ist. Das Meldungsfenster wird ausgeblendet, wenn `di5` auf 0 gesetzt wird oder wenn ein Timeout erfolgt (nach `60` Sekunden).

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `UIMsgBox` häufig, z. B. wie in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung führen, z.B. zu einer langsamen Reaktion des FlexPendant.

Syntax

```
UIMsgBox
  ['\'Header':=' <expression (IN) of string>',']
  [MsgLine1':='] <expression (IN) of string>
  ['\'MsgLine2':='<expression (IN) of string>]
  ['\'MsgLine3':='<expression (IN) of string>]
  ['\'MsgLine4':='<expression (IN) of string>]
  ['\'MsgLine5':='<expression (IN) of string>]
  ['\'Wrap]
  ['\'Buttons':=' <expression (IN) of buttondata>]
  ['\'Icon':=' <expression (IN) of icondata>]
  ['\'Image':='<expression (IN) of string>]
  ['\'Result':='< var or pers (INOUT) of btnres>]
  ['\'MaxTime':=' <expression (IN) of num>]
  ['\'DIBreak':=' <variable (VAR) of signaldi>]
  ['\'DIPassive]
  ['\'DOBreak':=' <variable (VAR) of signaldo>]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
['\'DOPassive]
['\'PersBoolBreak ':=' <persistent (PERS) of bool>]
['\'PersBoolPassive]
['\'BreakFlag':=' <var or pers (INOUT) of errnum>]
['\'UIActiveSignal ':=' <variable (VAR) of signaldo>] ';'
```

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Drucktastendaten	*buttondata – Drucktastendaten auf Seite 1642*
Ergebnisdaten für Drucktaste	*btnres - Ergebnisdaten für Drucktaste auf Seite 1639*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion "Zahleneingabe"	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion "Zahlensynchronisierung"	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion "Alpha-Eingabe"	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion "Listenansicht"	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
FlexPendant Interface	*Produktspezifikation - IRC5-Steuerung*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.318 UIMsgWrite - Benutzer-Meldungsfenster, Typ nicht-wartend

Verwendung

`UIMsgWrite` (*User Interaction Message Write*) wird verwendet, um mit dem Benutzer des Robotersystems über ein verfügbares Benutzergerät wie das FlexPendant zu kommunizieren. An den Bediener wird eine Meldung ausgegeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `UIMsgWrite`.

Beispiel 1:

```
VAR string myHeader := "Signal error!";

UIMsgWrite myHeader, "Set signal di1 high please!" \Icon:=iconInfo;
WaitDI di1, 1;
UIMsgWriteAbort;
```

Die Meldung "*Set signal di1 high please!*" wird angezeigt. Das Programm wird fortgesetzt und die Meldung wird entfernt, wenn das Signal `di1` gesetzt wird.

Beispiel 2:

```
VAR string myHeader := "Signal Error!";
VAR string myMsgArray{5}:= ["Set", "signal", "di1", "high",
     "please!"];
UIMsgWrite myHeader, myMsgArray, \Icon:=iconInfo
     \Image:="MyImage.jpg";
WaitDI di1, 1;
UIMsgWriteAbort;
```

Die Meldung mit dem Header, fünf Meldungszeilen, Icon und Bild wird angezeigt. Das Programm wird fortgesetzt und die Meldung wird entfernt, wenn das Signal `di1` gesetzt wird.

Beachten Sie, dass die Meldungszeilen 1 bis 5 in separaten Zeilen angezeigt werden (`\Wrap` wird nicht verwendet).

Argumente

```
UIMsgWrite Header Message | MsgArray [\Wrap] [\Icon] [\Image]
     [\PersBool] | [\PersBoolName] [\AbortValue] [\UIActiveSignal]
```

`Header`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`Message`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Maximal 50 Zeichen.

`MsgArray`

(Message Array)

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `string`

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter `Message` oder `MsgArray` verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Bei Auswahl werden die Zeilen in `MsgArray` zu einer Zeichenfolge zusammengefasst und auf so wenig Zeilen wie möglich verteilt, wobei die einzelnen Zeichenfolgen nur durch ein einzelnes Leerzeichen voneinander getrennt werden.

Standardmäßig wird jede Zeile in `MsgArray` auf dem Display in einer separaten Zeile angezeigt.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs icondata verwendet werden, siehe *Vordefinierte Daten auf Seite 1023*.

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

`[\PersBool]`

(Persistent Bool)

Datentyp: `bool`

Die Meldung wird so lange angezeigt, wie die angegebene `bool` den Wert `FALSE` hat. Wenn der Parameter `AbortValue` verwendet wird, wird die Meldung so lange angezeigt, wie die angegebene `bool` einen anderen Wert als `AbortValue` hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.318 UIMsgWrite - Benutzer-Meldungsfenster, Typ nicht-wartend
RobotWare - OS
Fortsetzung

Es kann nur eine vollständige PERS- oder TASK PERS Bool-Variable verwendet werden.

`[\PersBoolName]`

(Persistent Bool Name)

Datentyp: `string`

Die Meldung wird so lange angezeigt, wie die angegebene `bool` den Wert `FALSE` hat. Wenn der Parameter `AbortValue` verwendet wird, wird die Meldung so lange angezeigt, wie die angegebene `bool` einen anderen Wert als `AbortValue` hat.

Es kann nur ein vollständiger PERS- oder TASK PERS Bool-Variablenname verwendet werden.

Wenn das Argument `\PersBoolName` verwendet wird, ist es möglich, eine PERS-Bool-Variable zu verwenden, die in einer anderen Task in der Instruktion `UIMsgWrite` deklariert wird.

`[\AbortValue]`

Datentyp: `bool`

Gilt nur, wenn `PersBool` vorhanden ist. Der erwartete Wert für `PersBool`. Die Meldung wird angezeigt, bis `PersBool` dem Wert `AbortValue` entspricht.

`[\UIActiveSignal]`

Datentyp: `signaldo`

Das digitale Ausgangssignal, das im optionalen Argument `UIActiveSignal` verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wird. Das Signal wird auf 0 gesetzt, wenn die Meldungsbox durch Instruktion `UIMsgWriteAbort` entfernt wird oder wenn der Ausdruck `PersBool` erfüllt ist.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn der PP bewegt wird.

Programmabarbeitung

Die Meldungsbox mit dem Icon, dem Header, den Meldungszeilen, dem Bild und den nicht erfüllten Ausdrücken wird gemäß den programmierten Argumenten angezeigt. Die Meldung wird angezeigt, bis sie durch `UIMsgWriteAbort` abgebrochen wird (oder wenn der Ausdruck `PersBool` erfüllt ist, wenn das Argument `PersBool` verwendet wird).

- Eine neue Meldung auf Basisebene ersetzt eine ältere Nachricht.
- Eine neue Meldung auf TRAP-Ebene ersetzt eine ältere Meldung auf Basisebene und bleibt aktiv, wenn zur Basisebene zurückgekehrt wird.
- Eine neue Nachricht in einer Serviceroutine wird immer abgebrochen, wenn zur Basisebene zurückgekehrt wird. Eine aktive Meldung auf Basisebene wird dann erneut aktiviert.
- Eine neue Benutzermeldung, die durch die folgenden Instruktionen und Funktionen erzeugt wird, ersetzt eine Meldung, die durch `UIMsgWrite` erzeugt wurde:

  `TPReadFK, TPReadDnum, TPReadNum, UIMsgBox, UIMessageBox, UIDnumEntry, UIDnumTune, UINumEntry, UINumTune,`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`UIAlphaEntry`, `UIListView`. Ein Meldungsfentser von einer der ausstehenden Instruktionen `WaitAI`, `WaitAO`, `WaitGI`, `WaitGO`, `WaitDI`, `WaitDO`, `WaitUntil` ersetzt eine von `UIMsgWrite` generierte Meldung.

Vordefinierte Daten

Folgende Konstanten des Datentyps `icondata` sind im System vordefiniert:

Wert	Konstante	Symbol
0	`iconNone`	Kein Icon
1	`iconInfo`	Informations-Icon
2	`iconWarning`	Warn-Icon
3	`iconError`	Fehler-Icon

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
`ERR_SYM_ACCESS`	Das in `[\PersBoolName]` verwendete Datenobjekt existiert nicht.

Syntax

```
UIMsgWrite
  [Header ':=' <expression (IN) of string> ',']
  [Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' PersBool ':=' <pers (IN) of bool>]
  | ['\' PersBoolName ':=' <pers (IN) of string>]
  ['\' AbortValue ':=' <var or pers (IN) of bool>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ';'
```

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.318 UIMsgWrite - Benutzer-Meldungsfenster, Typ nicht-wartend

RobotWare - OS

Fortsetzung

Information über	Siehe
Dialog "Abbruch Benutzermeldung"	*UIMsgWriteAbort - Benutzer-Meldungsfenster, Typ nicht-wartend, stornieren auf Seite 1025*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
FlexPendant Interface	*Produktspezifikation - IRC5-Steuerung*
Cyclic bool	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.319 UIMsgWriteAbort - Benutzer-Meldungsfenster, Typ nicht-wartend, stornieren

Verwendung

`UIMsgWriteAbort` wird verwendet, um eine aktive Meldung zu stornieren, die zuvor durch eine `UIMsgWrite`-Instruktion erzeugt wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `UIMsgWriteAbort`.

Beispiel 1:

```
VAR string myHeader := "Signal error!";

UIMsgWrite myHeader, "Set signal di1 high please!", iconInfo;
WaitDI di1, 1;
UIMsgWriteAbort;
```

Die Meldung "*Set signal di1 high please!*" wird angezeigt. Das Programm wird fortgesetzt und die Meldung wird entfernt, wenn das Signal `di1` gesetzt wird.

Syntax

```
UIMsgWriteAbort ';'
```

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Dialog "Benutzermeldung schreiben"	*UIMsgWrite - Benutzer-Meldungsfenster, Typ nicht-wartend auf Seite 1020*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
FlexPendant Interface	*Produktspezifikation - IRC5-Steuerung*
Cyclic bool	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.320 UIShow - Anzeigen der Benutzeroberfläche

Verwendung

UIShow (*User Interface Show*) wird verwendet, um mit dem Benutzer des Robotersystems über das verfügbare Benutzergerät, z. B. das FlexPendant, zu kommunizieren. Mit UIShow können individuell geschriebene Anwendungen und Standardanwendungen aus einem RAPID-Programm gestartet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion UIShow:

Beispiel 1 und 2 können nur ausgeführt werden, wenn die Dateien TpsViewMyAppl.dll und TpsViewMyAppl.gtpu.dll im Verzeichnis HOME: vorhanden sind und ein **Warmstart** ausgeführt wurde.

Beispiel 1:

```
CONST string Name:="TpsViewMyAppl.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.TpsViewMyAppl";
CONST string Cmd1:="Init data string passed to the view";
CONST string Cmd2:="New init data string passed to the view";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
! Launch one view of my application MyAppl
UIShow Name, Type \InitCmd:=Cmd1 \InstanceID:=myinstance
     \Status:=mystatus;
! Update the view with new init command
UIShow Name, Type \InitCmd:=Cmd2 \InstanceID:=myinstance
     \Status:=mystatus;
```

Der oben dargestellte Code öffnet die Ansicht TpsViewMyAppl mit dem Initialisierungsbefehl Cmd1 und aktualisiert anschließend die Ansicht mit Cmd2.

Beispiel 2:

```
CONST string Name:="TpsViewMyAppl.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.TpsViewMyAppl";
CONST string Cmd1:="Init data string passed to the view";
CONST string Cmd2:="New init data string passed to the view";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
! Launch one view of my application MyAppl
UIShow Name, Type \InitCmd:=Cmd1 \Status:=mystatus;
! Launch another view of the application MyAppl
UIShow Name, Type \InitCmd:=Cmd2 \InstanceID:=myinstance
     \Status:=mystatus;
```

Der oben dargestellte Code öffnet die Ansicht TpsViewMyAppl mit dem Initialisierungsbefehl Cmd1. Anschließend wird mit dem Initialisierungsbefehl Cmd2 eine weitere Ansicht geöffnet.

Beispiel 3

```
CONST string Name:="tpsviewbackupandrestore.dll";
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
CONST string Type:="ABB.Robotics.Tps.Views.TpsViewBackupAndRestore";
VAR num mystatus:=0;
...
UIShow Name, Type \Status:=mystatus;
```

Startet die Standardanwendung Backup and Restore.

Beispiel 4:

```
CONST string Name:="TpsViewPanel.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.MainScreen";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
UIShow Name, Type \InstanceID:=myinstance \Status:=mystatus;
```

Starten Sie eine mit `ScreenMaker` erstellte Anwendung.

Argumente

```
UIShow AssemblyName TypeName [\InitCmd] [\InstanceId] [\Status]
     [\NoCloseBtn]
```

`AssemblyName`

Datentyp: `string`

Der Name der Assembly, die die Ansicht enthält.

`TypeName`

Datentyp: `string`

Dies ist der Name der Ansicht (der zu erstellende Typ). Dabei handelt es sich um den vollqualifizierten Namen des Typs, d. h., der Namespace ist darin enthalten.

`[\InitCmd]`

Init Command

Datentyp: `string`

Eine Zeichenfolge mit Initialisierungsdaten, die an die Ansicht übergeben wird.

`[\InstanceId]`

Datentyp: `uishownum`

Ein Parameter, der in Token zum Identifizieren der Ansicht darstellt. Wenn eine Ansicht nach dem Aufruf von `UIShow` angezeigt wird, wird ein Wert, der die Ansicht identifiziert, zurückgegeben. Dieses Token kann anschließend in anderen Aufrufen von `UIShow` verwendet werden, um eine gegenwärtig ausgeführte Ansicht zu aktivieren. Wenn der Wert eine vorhandene (gerade ausgeführte) Ansicht identifiziert, wird die Ansicht aktiviert. Wenn die Ansicht nicht vorhanden ist, wird eine neue Instanz erstellt. Daher kann mit diesem Parameter bestimmt werden, ob eine neue Instanz gestartet wird. Wenn sein Wert eine bereits gestartete Ansicht identifiziert, wird diese Ansicht aktiviert, unabhängig von den Werten aller anderen Parameter. Es wird empfohlen, für jede mit der `UIShow`-Instruktion zu startende neue Anwendung eine eindeutige `InstanceId`-Variable zu verwenden.

Der Parameter muss eine persistente Variable sein. Der Grund hierfür ist, dass der Wert dieser Variablen beibehalten werden muss, auch wenn der Programmzeiger nach „main“ verschoben wird. Wenn dieselbe `UIShow`-Instruktion

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wie zuvor mit derselben Variablen ausgeführt wird, wird dieselbe Ansicht aktiviert, sofern sie noch geöffnet ist. Wenn die Ansicht geschlossen wurde, wird eine neue Ansicht gestartet.

`[\Status]`

Datentyp: `num`

`Status` gibt an, ob der Vorgang erfolgreich ausgeführt wurde. Bei Verwendung dieser Option wartet die RAPID-Abarbeitung bis zum Abschluss der Instruktion, d. h. bis die Ansicht gestartet wurde.

Dieser optionale Parameter wird hauptsächlich zum Debuggen verwendet. (Siehe *Fehlerbehebung*)

Status	Beschreibung
0	OK
-1	Auf dem FlexPendant ist kein Arbeitsspeicher für die neue Ansicht vorhanden. Auf dem FlexPendant können höchstens 6 Ansichten gleichzeitig geöffnet sein.
-2	Die Assembly wurde nicht gefunden, ist nicht vorhanden.
-3	Dei Datei wurde gefunden, konnte jedoch nicht geladen werden.
-4	Die Assembly ist vorhanden, es konnte jedoch keine neue Instanz erstellt werden.
-5	Der typename ist für diese Assembly ungültig.
-6	InstanceID stimmt nicht mit der zu ladenden Assembly überein.

`[\NoCloseBtn]`

No Close Button

Datentyp: `switch`

`NoCloseBtn` deaktiviert die Schaltfläche zum Schließen der Ansicht.

Programmabarbeitung

Die `UIShow`-Instruktion wird verwendet, um individuelle Anwendungen auf dem FlexPendant zu starten. Um individuelle Anwendungen zu starten, müssen die Assembly-Dateien im Verzeichnis `HOME:` des aktiven Systems oder direkt im aktiven System oder einer zusätzlichen Option abgelegt werden. Es wird empfohlen, die Dateien im Verzeichnis `HOME:` abzulegen, so dass sie bei einem Backup und Restore gespeichert werden. Ein **Neustart** muss durchgeführt werden. Anschließend lädt das FlexPendant die neuen Baugruppen.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Es können auch Standardanwendungen, z. B. Backup and Restore, gestartet werden. Dann muss die RobotWare-Option *FlexPendant Interface* nicht vorhanden sein.

Bei Verwendung des Arguments `\Status` wird das Programm erst nach dem Starten der Anwendung abgearbeitet. Wenn Fehler in der Anwendung nicht behandelt werden, wird nur das Ergebnis des Startvorgangs überwacht. Wenn der `\Status`-Parameter nicht vorhanden ist, erhält das FlexPendant den Befehl zum

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Starten der Anwendung, doch es wird nicht überprüft, ob die Anwendung gestartet werden kann.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.

Wenn der Parameter \Status verwendet wird, können folgende Situationen in der Fehlerbehandlungsroutine behandelt werden:

- Wenn auf dem FlexPendant kein Speicherplatz für die Assembly vorhanden ist, wird die Systemvariable ERRNO auf ERR_UISHOW_FULL gesetzt und die Abarbeitung wird in der Fehlerbehandlungsroutine fortgesetzt. Auf dem FlexPendant können 6 Ansichten gleichzeitig geöffnet sein.
- Wenn beim Starten der Ansicht ein anderer Fehler auftritt, wird die Systemvariable ERRNO auf ERR_UISHOW_FATAL gesetzt und die Abarbeitung wird in der Fehlerbehandlungsroutine fortgesetzt.

Einschränkungen

Wenn individuelle Anwendungen mithilfe der Instruktion UIShow gestartet werden, muss das System über die Option *FlexPendant Interface* verfügen.

Anwendungen, die mit der Instruktion UIShow gestartet wurden, werden bei einem Stromausfall nicht weiter ausgeführt. Mit der POWER ON-Ereignisroutine kann die Anwendung erneut eingerichtet werden.

Syntax

```
UIShow
  [AssemblyName ':='] < expression (IN) of string >','
  [TypeName ':='] < expression (IN) of string >','
  ['\'InitCmd' :=' < expression (IN) of string> ]
  ['\'InstanceId ':=' < persistent (PERS) of uishownum> ]
  ['\'Status ':=' < variable (VAR) of num> ]
  ['\'NoCloseBtn ]';'
```

Weitere Informationen

Information über	Siehe
FlexPendant interface	*Produktspezifikation - IRC5-Steuerung*
Erstellen individueller Anwendungen für das FlexPendant	*http://developercenter.robotstudio.com/*
uishownum	*uishownum - Instanz-ID für UIShow auf Seite 1849*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.321 UnLoad - Entladen eines Programmmoduls während der Abarbeitung

Verwendung

`UnLoad` wird verwendet, um ein Programmmodul während der Abarbeitung aus dem Programmspeicher zu entfernen.

Das Programmmodul muss zuvor über die Instruktion `Load` oder `StartLoad - WaitLoad` in den Programmspeicher geladen worden sein.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `UnLoad`:

Siehe auch *Weitere Beispiele auf Seite 1031* unten.

Beispiel 1:

```
UnLoad diskhome \File:="PART_A.MOD";
```

Mit `UnLoad` wird das Programmmodul PART_A.MOD entladen, das zuvor mit `Load` in den Programmspeicher geladen wurde. (Siehe Instruktion `Load`). `diskhome` ist eine vordefinierte string-Konstante für "HOME:".

Argumente

```
UnLoad [\ErrIfChanged] | [\Save] FilePath [\File]
```

`[\ErrIfChanged]`

Datentyp: `switch`

Wenn dieses Argument verwendet wird und das Modul geändert wurde, nachdem es ins System geladen wurde, generiert die Instruktion den Fehlercode `ERR_NOTSAVED`.

`[\Save]`

Datentyp: `switch`

Wenn dieses Argument verwendet wird, wird das Programmmodul gespeichert, bevor es entladen wird. Das Programmmodul wird an dem Ort gespeichert, der in der Instruktion `Load` oder `StartLoad` angegeben ist.

`FilePath`

Datentyp: `string`

Pfad und Name der Datei, die aus dem Programmspeicher entfernt wird. Pfad und Name der Datei müssen gleich sein wie in der zuvor abgearbeiteten Instruktion `Load` oder `StartLoad`. Der Dateiname soll ausgeschlossen werden, wenn das Argument `\File` verwendet wird.

`[\File]`

Datentyp: `string`

Wenn der Dateiname im Argument `FilePath` ausgelassen wird, muss er mit diesem Argument definiert werden. Der Dateiname muss gleich sein wie in der zuvor abgearbeiteten Instruktion `Load` oder `StartLoad` .

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Um die Instruktion `UnLoad` in einem Programm abzuarbeiten, muss die Instruktion `Load` oder `StartLoad - WaitLoad` mit demselben Dateipfad und demselben Dateinamen an einer früheren Stelle im Programm abgearbeitet worden sein.

Die Programmabarbeitung wartet das Ende des Entladevorgangs ab, bevor sie mit der nächsten Instruktion fortfährt.

Nachdem das Programmmodul entladen wurde, werden die übrigen Programmmodule verbunden.

Weitere Informationen finden Sie unter den Instruktionen `Load` oder `StartLoad-Waitload`.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NOTSAVED`	Das Argument `\ErrIfChanged` wird verwendet und das Modul wurde geändert.
`ERR_UNLOAD`	Die in der Instruktion `UnLoad` angegebene Datei kann wegen einer laufenden Abarbeitung innerhalb des Moduls oder einer falschen Pfadangabe nicht entladen werden (Modul nicht mit `Load` oder `StartLoad` geladen).

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `UnLoad` sind unten aufgeführt.

Beispiel 1:

```
UnLoad "HOME:/DOORDIR/DOOR1.MOD";
```

Mit `UnLoad` wird das Programmmodul `DOOR1.MOD` entladen, das zuvor in den Programmspeicher geladen wurde.

Beispiel 2:

```
UnLoad "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Wie in Beispiel 1, aber unterschiedliche Syntax.

Beispiel 3

```
Unload \Save, "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Wie in den Beispielen 1 und 2 oben, aber das Programmmodul wird vor dem Entladen gespeichert.

Einschränkungen

Es ist nicht erlaubt, ein Programmmodul zu entladen, das gerade abgearbeitet wird (der Programmzeiger befindet sich im Modul).

Interrupt-Routinen, E/A-Systemereignisse und andere Programmtasks können während des Entladevorgangs nicht abgearbeitet werden.

Verhindern Sie Roboterbewegungen während des Entladevorgangs.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.321 UnLoad - Entladen eines Programmmoduls während der Abarbeitung

RobotWare - OS

Fortsetzung

Ein Programmstopp während der Abarbeitung der Instruktion `UnLoad` kann zu einem Sicherheitsstopp mit ausgeschalteten Motoren führen und die Fehlermeldung "20025 Stopp-Anweisung Timeout" wird am FlexPendant angezeigt.

Syntax

```
UnLoad
  ['\'ErrIfChanged ','] | ['\'Save ','']
  [FilePath':=']<expression (IN) of string>
  ['\'File':=' <expression (IN) of string>]';'
```

Weitere Informationen

Information über	Siehe
Überprüft Programmreferenzen.	*CheckProgRef - Überprüfen von Programmverweisen auf Seite 131*
Laden eines Programmmoduls	*Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346* *StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802* *WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.322 UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten

Verwendung

`UnpackRawBytes` wird verwendet, um den Inhalt eines Containers vom Typ `rawbytes` in Variablen vom Typ `byte`, `num`, `dnum` oder `string` zu entpacken.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `UnpackRawBytes`.

Beispiel 1:

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num integer;
VAR dnum bigInt;
VAR num float;
VAR string string1;
VAR byte byte1;
VAR byte data1;

! Data packed in raw_data_out according to the protocol
...
Open "chan1:", io_device\Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in\Time := 1;
Close io_device;
```

Laut Protokoll, das dem Programmierer bekannt ist, wird die Nachricht an das Gerät `"chan1:"` gesendet. Dann wird die Antwort vom Gerät abgelesen.

Die Antwort hat z. B. folgenden Inhalt:

Bytenummer:	Inhalt:
1-4	Ganzzahl 5
5-8	Gleitkommazahl 234.6
9-25	Zeichenfolge "This is real fun!". Es handelt sich um eine ISO 8859-1-Zeichfolge (Latin-1) mit Einzelbyte-Zeichen.
26	Hex-Wert '4D'
27	ASCII-Code 122, das heißt, z
28-36	Ganzzahl 4294967295
37-40	Ganzzahl 4294967295

```
UnpackRawBytes raw_data_in, 1, integer \IntX := DINT;
```

Der Inhalt von `integer` ist 5.

```
UnpackRawBytes raw_data_in, 5, float \Float4;
```

Der Inhalt von `float` ist 234.6, dezimal.

```
UnpackRawBytes raw_data_in, 9, string1 \ASCII:=17;
```

Der Inhalt von `string1` ist "This is real fun!".

```
UnpackRawBytes raw_data_in, 26, byte1 \Hex1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.322 UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten

RobotWare - OS

Fortsetzung

Der Inhalt von `byte1` ist `'4D'`, hexadezimal.

```
UnpackRawBytes raw_data_in, 27, data1 \ASCII:=1;
```

Der Inhalt von `data1` ist `122`, der ASCII-Code für "z".

```
UnpackRawBytes raw_data_in, 28, bigInt \IntX := LINT;
```

Der Inhalt von `bigInt` ist `4294967295`.

```
UnpackRawBytes raw_data_in, 37, bigInt \IntX := UDINT;
```

Der Inhalt von `bigInt` ist `4294967295`.

Argumente

```
UnpackRawBytes RawData [ \Network ] StartIndex Value [\Hex1 ] | [
    \IntX ] | [ \Float4 ] | [ \ASCII ]
```

Siehe *Argumente kombinieren auf Seite 1035*.

`RawData`

Datentyp: `rawbytes`

Variablencontainer, aus dem die Daten entpackt werden sollen

`[ \Network ]`

Datentyp: `switch`

Gibt an, dass die Ganzzahl und der Float-Wert aus der Big endian-Darstellung (Netzwerkreihenfolge) in `RawData` entpackt werden sollen. Big endian wird in ProfiBus und InterBus verwendet.

Ohne diesen Schalter werden die Ganzzahl und der Float-Wert aus der Little endian-Darstellung (keine Netzwerkreihenfolge) in `RawData` entpackt. Little-endian wird in DeviceNet verwendet.

Nur relevant gemeinsam mit dem Optionsparameter `\IntX - UINT`, `UDINT, , ULINT, INT, DINT, LINT` und `\Float4`.

`StartIndex`

Datentyp: `num`

`StartIndex`, zwischen 1 und 1024, gibt an, wo mit dem Entpacken aus `RawData` begonnen werden soll.

`Value`

Datentyp: `anytype`

Variable mit den Daten, die aus `RawData` entpackt wurden.

Mögliche Datentypen: `byte`, `num`, `dnum` oder `string`. Datenfelder können nicht verwendet werden.

`[ \Hex1 ]`

Datentyp: `switch`

Daten, die entpackt und in `Value` verschoben werden sollen, sind in 1 `byte` im Hexadezimalformat geschrieben und werden in einer `byte`-Variablen ins Dezimalsystem umgerechnet.

`[ \IntX ]`

Datentyp: `inttypes`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Daten, die entpackt werden sollen, haben das Format laut angegebener Konstante des Datentyps `inttypes`. Die Daten werden in eine `num` oder eine `dnum`-Variable konvertiert, die eine Ganzzahl enthält und die in `Value` gespeichert ist.

Siehe *Vordefinierte Daten auf Seite 1036*.

`[ \Float4 ]`

Datentyp: `switch`

Die Daten, die entpackt und in Value eingesetzt werden sollen, haben das Format Float, 4 Bytes, und werden in eine `num`-Variable konvertiert, die einen Float-Wert enthält.

`[ \ASCII ]`

Datentyp: `num`

Die Daten, die entpackt und in `Value` eingesetzt werden sollen, haben das Format `byte` oder `string`.

Wenn es sich bei `Value` um den Typ `byte` handelt, werden die Daten als ASCII-Code interpretiert und in das `byte`-Format konvertiert (1 Zeichen).

Wenn es sich bei `Value` um den Typ `string` handelt, werden die Daten als `string` gespeichert (1-80 Zeichen). String-Daten sind in Daten des Typs `rawbytes` nicht null-terminiert.

Argumente kombinieren

Eines der Argumente `\Hex1`, `\IntX`, `\Float4` oder `\ASCII` muss verwendet werden.

Die folgenden Kombinationen sind möglich:

Datentyp `Value`:	Mögliche optionale Parameter:
`num` [i]	`\IntX`
`dnum` [ii]	`\IntX`
`num`	`\Float4`
`string`	`\ASCII` (1 bis 80 Zeichen)
`byte`	`\Hex1 \ASCII`

[i] Muss eine Ganzzahl sein innerhalb des Wertebereichs der ausgewählten symbolischen Konstante `USINT`, `UINT`, `UDINT`, `SINT`, `INT` oder `DINT`.

[ii] Muss eine Ganzzahl sein innerhalb des Wertebereichs der ausgewählten symbolischen Konstante `USINT`, `UINT`, `UDINT`, `ULINT`, `SINT`, `INT`, `DINT` oder `LINT`.

Programmabarbeitung

Bei der Programmabarbeitung werden Daten aus dem "Container" des Typs `rawbytes` in eine Variable des Typs `anytype` entpackt.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.322 UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten

RobotWare - OS

Fortsetzung

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `inttypes` sind vordefiniert und können zur Angabe der Ganzzahl im Parameter `\IntX` verwendet werden.

Symbolische Konstante	Konstanter Wert	Ganzzahlformat	Wertebereich der Ganzzahl
`USINT`	1	Vorzeichenlose 1-Byte-Ganzzahl	0 ... 255
`UINT`	2	Vorzeichenlose 2-Byte-Ganzzahl	0 ... 65 535
`UDINT`	4	Vorzeichenlose 4-Byte-Ganzzahl	0 ... 8 388 608 [i] 0 ... 4 294 967 295 [ii]
`ULINT`	8	Vorzeichenlose 8-Byte-Ganzzahl	0 ... 4 503 599 627 370 496 [iii]
`SINT`	- 1	Vorzeichenbehaftete 1-Byte-Ganzzahl	- 128... 127
`INT`	- 2	Vorzeichenbehaftete 2-Byte-Ganzzahl	- 32 768 ... 32 767
`DINT`	- 4	Vorzeichenbehaftete 4-Byte-Ganzzahl	- 8 388 607 ... 8 388 608 [i] -2 147 483 648 ... 2 147 483 647 [iv]
`LINT`	- 8	Vorzeichenbehaftete 8-Byte-Ganzzahl	- 4 503 599 627 370 496... 4 503 599 627 370 496 [iii]

i Begrenzung in RAPID für die Speicherung einer Ganzzahl vom Datentyp `num`.
ii Bereich bei Verwendung einer dnum-Variablen und inttype `UDINT`.
iii Begrenzung in RAPID für die Speicherung einer Ganzzahl vom Datentyp `dnum`.
iv Bereich bei Verwendung einer dnum-Variablen und inttype `DINT`.

Syntax

```
UnpackRawBytes
  [RawData ':='] <variable (VAR) of rawbytes>
  ['\' Network]','
  [StartIndex ':='] <expression (IN) of num>','
  [Value ':='] <variable (VAR) of anytype>
  ['\' Hex1]
  |['\' IntX' :=' <expression (IN) of inttypes>]
  |['\' Float4 ]
  |['\' ASCII' :=' <expression (IN) of num>]';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Bit/Byte-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.323 VelSet - Ändern der programmierten Geschwindigkeit

Verwendung

`VelSet` wird verwendet, um die programmierte Geschwindigkeit aller nachfolgenden Bewegungsinstruktionen zu erhöhen oder zu verringern. Diese Instruktion dient auch der Begrenzung der maximalen TCP-Geschwindigkeit.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `VelSet`:

Siehe auch *Weitere Beispiele auf Seite 1039*.

Beispiel 1:

```
VelSet 50, 800;
```

Alle programmierten Geschwindigkeiten werden in dieser Instruktion auf 50 % ihres Werts reduziert. Die TCP-Geschwindigkeit darf jedoch nicht 800 mm/s überschreiten.

Argumente

```
VelSet Override Max
```

`Override`

Datentyp: `num`

Gewünschte Geschwindigkeit als Prozentsatz der programmierten Geschwindigkeit. 100 % entspricht der programmierten Geschwindigkeit.

`Max`

Datentyp: `num`

TCP-Höchstgeschwindigkeit in mm/s

Programmabarbeitung

Die programmierte Geschwindigkeit gilt für die nächste abgearbeitete Bewegungsinstruktion, bis eine neue `VelSet`-Instruktion ausgeführt wird.

Das Argument `Override` beeinflusst:

- alle Geschwindigkeitskomponenten (TCP, Orientierung, Rotation und lineare externe Achsen) in `speeddata`.
- den programmierten Geschwindigkeitsoverride in der Positionierungsinstruktion (das Argument `\V`).
- zeitgesteuerte Bewegungen.

Das Argument `Override` hat keinen Einfluss auf:

- die Schweißgeschwindigkeit in `welddata`.
- die Wärm- und Füllgeschwindigkeit in `seamdata`.

Das Argument `Max` begrenzt die Geschwindigkeit des TCP nur, wenn sie niedriger als die programmierte Geschwindigkeit ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Standardwerte für `Override` und `Max` sind 100% bzw. `vmax.v_tcp` mm/s. Diese Werte werden automatisch eingestellt.

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Hinweis

Die maximale TCP-Geschwindigkeit für den verwendeten Robotertyp kann in der Konfiguration des Systemparameters *Motion*, Typ *Motion Planner* und Attribut *Linear Max Speed*, geändert werden. Die RAPID-Funktion `MaxRobSpeed` gibt denselben Wert zurück.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `VelSet` sind unten aufgeführt.

Beispiel 1:

```
VelSet 50, 800;
MoveL p1, v1000, z10, tool1;
MoveL p2, v2000, z10, tool1;
MoveL p3, v1000\T:=5, z10, tool1;
```

Die Geschwindigkeit beträgt 500 mm/s zu Punkt `p1` und 800 mm/s zu `p2`. Die Bewegung von `p2` zu `p3` dauert 10 Sekunden.

Einschränkungen

Die Höchstgeschwindigkeit wird nicht berücksichtigt, wenn die Zeit in den Bewegungsinstruktionen angegeben wird.

Syntax

```
VelSet
  [ Override ':=' ] < expression (IN) of num > ','
  [ Max ':=' ] < expression (IN) of num > ';'
```

Weitere Informationen

Information über	Siehe
Reduktion der Beschleunigung	*AccSet - Verringern der Beschleunigung auf Seite 25*
TCP-Höchstgeschwindigkeit für diesen Roboter	*MaxRobSpeed - Höchstgeschwindigkeit des Roboters auf Seite 1394*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Definition der Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.324 WaitAI - Warten, bis ein analoger Eingangssignalwert gesetzt wird

Verwendung

`WaitAI` (*Wait Analog Input*) wird verwendet, um zu warten, bis ein analoger Eingangssignalwert gesetzt wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WaitAI`:

Beispiel 1:

```
WaitAI ai1, \GT, 5;
```

Die Programmabarbeitung fährt erst fort, nachdem der analoge Eingang `ai1` einen größeren Wert als 5 aufweist.

Beispiel 2:

```
WaitAI ai1, \LT, 5;
```

Die Programmabarbeitung fährt erst fort, nachdem der analoge Eingang `ai1` einen kleineren Wert als 5 aufweist.

Argumente

```
WaitAI Signal [\LT] | [\GT] Value [\MaxTime] [\ValueAtTimeout]
     [\Visualize] [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon] [\Image] [\VisualizeTime] [\UIActiveSignal]
     [\ErrorNumber]
```

`Signal`

Datentyp: `signalai`

Name des analogen Eingangssignals.

`[\LT]`

Less Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitAI`, bis der Wert des analogen Signals kleiner als `Value` ist.

`[\GT]`

Greater Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitAI`, bis der Wert des analogen Signals größer als `Value` ist.

`Value`

Datentyp: `num`

Der gewünschte Wert des Signals.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode ERR_WAIT_MAXTIME aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

[\ValueAtTimeout]

Datentyp: num

Bei einem Timeout der Instruktion wird der aktuelle Signalwert in dieser Variablen gespeichert. Die Variable wird nur gesetzt, wenn die Systemvariable ERRNO auf ERR_WAIT_MAXTIME gesetzt ist.

[\Visualize]

Datentyp: switch

Wenn ausgewählt, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der Bedingung, die nicht erfüllt ist, aus Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

[\Header]

Datentyp: string

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein\Header Argument eingegeben wird, wird eine Standardnachricht angezeigt.

[\Message]

Datentyp: string

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

[\MsgArray]

(Message Array)

Datentyp: string

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter \Message oder \MsgArray verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

[\Wrap]

Datentyp: switch

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument \MsgArray zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument \MsgArray in einer eigenen Zeile auf dem Display angezeigt.

[\Icon]

Datentyp: icondata

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs icondata verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.324 WaitAI - Warten, bis ein analoger Eingangssignalwert gesetzt wird
RobotWare - OS
Fortsetzung

Standardmäßig wird kein Icon angezeigt.

[\Image]

Datentyp: string

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

[\VisualizeTime]

Datentyp: num

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente \VisualizeTime und \MaxTime verwendet werden, muss die im Argument \MaxTime verwendete Zeit größer sein als die Zeit, die im Argument \VisualizeTime verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument \VisualizeTime ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignalverwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

[\ErrorNumber]

Error number

Datentyp: errnum

Eine Variable (vor Verwendung vom System auf 0 gesetzt), die den Fehlercode enthält, wenn die Anweisung endet, bevor das Signal den gewünschten Wert hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn diese optionale Variable weggelassen wird, dann wird die Fehlerbehandlung ausgeführt. Die Konstanten ERR_AO_LIM, ERR_NO_ALIASIO_DEF, ERR_NORUNUNIT, ERR_SIG_NOT_VALID und ERR_WAIT_MAXTIME können für die Auswahl der Begründung verwendet werden.

Programmabarbeitung

Wenn der Wert des Signals beim Abarbeiten der Instruktion korrekt ist, fährt das Programm einfach mit der nachfolgenden Instruktion fort.

Wenn der Signalwert nicht korrekt ist, geht der Roboter in einen Wartestatus über, und sobald das Signal den korrekten Wert annimmt, fährt das Programm fort. Die Änderung wird mit einem Interrupt ermittelt, der schnell reagiert (nicht abgefragt wird).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem optionalen Argument \MaxTime angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Wenn die Programmabarbeitung angehalten wurde und später neu gestartet wird, wertet die Instruktion den aktuellen Wert des Signals aus. Jede Änderung während des Programmstopps wird zurückgewiesen.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter \Visualize verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein \Header-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der WaitAI-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte Value-Argument für das angegebene analoge Eingangssignal Signal liegt außerhalb des zulässigen Bereichs.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.324 WaitAI - Warten, bis ein analoger Eingangssignalwert gesetzt wird
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
ERR_WAIT_MAXTIME	Es liegt ein Timeout vor (Parameter \MaxTime), bevor das Signal zum richtigen Wert wechselt.

Weitere Beispiele

Weitere Beispiele für die Instruktion WaitAI sind unten aufgeführt.

Beispiel 1:

```
VAR num myvalattimeout:=0;
WaitAI ai1, \LT, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of ai1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Die Programmabarbeitung wird nur fortgesetzt, wenn ai1 kleiner als 5 ist oder wenn ein Timeout erfolgt. Bei einem Timeout kann der Wert des Signals ai1 zum Zeitpunkt des Timeouts ohne ein weiteres Lesen des Signals protokolliert werden.

Beispiel 2:

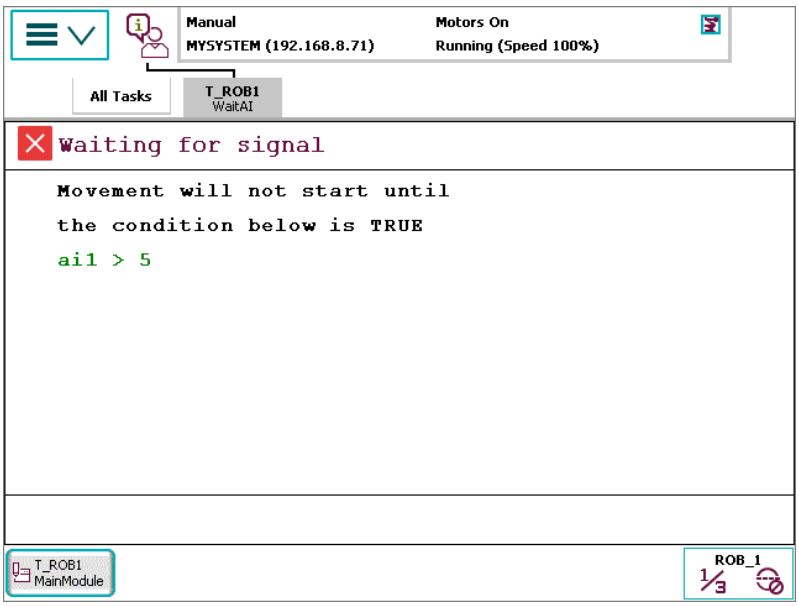
```
WaitAI ai1 \GT, 5 \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn diese Vorgabe nicht erfüllt wird, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen \Header wiedergegeben und \MsgArray wird auf

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

der Anzeige des FlexPendant gemeinsam mit der Bedingung, die nicht erfüllt wurde, angezeigt.



xx1600000150

Syntax

```
WaitAI
  [ Signal ':=' ] <variable (VAR) of signalai> ','
  [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] <expression (IN) of num>
  ['\'MaxTime ':='<expression (IN) of num>]
  ['\'ValueAtTimeout' :=' <variable (VAR) of num>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>]
  ['\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
       ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Voraussetzung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*
Warten, bis eine vorgegebene Dauer verstrichen ist	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*
Warten, bis ein analoger Ausgang gesetzt/zurückgesetzt wird	*WaitAO - Warten, bis ein analoger Ausgangssignalwert gesetzt wird auf Seite 1046*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.325 WaitAO - Warten, bis ein analoger Ausgangssignalwert gesetzt wird

Verwendung

`WaitAO` (*Wait Analog Output*) wird verwendet, um zu warten, bis ein analoger Ausgangssignalwert gesetzt wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WaitAO`:

Beispiel 1:

```
WaitAO ao1, \GT, 5;
```

Die Programmabarbeitung fährt erst fort, nachdem der analoge Ausgang `ao1` einen größeren Wert als 5 aufweist.

Beispiel 2:

```
WaitAO ao1, \LT, 5;
```

Die Programmabarbeitung fährt erst fort, nachdem der analoge Ausgang `ao1` einen kleineren Wert als 5 aufweist.

Argumente

```
WaitAO Signal [\LT] | [\GT] Value [\MaxTime] [\ValueAtTimeout]
     [\Visualize] [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon] [\Image] [\VisualizeTime] [\UIActiveSignal]
     [\ErrorNumber]
```

`Signal`

Datentyp: `signalao`

Name des analogen Ausgangssignals

`[\LT]`

Less Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitAO`, bis der Wert des analogen Signals kleiner als `Value` ist.

`[\GT]`

Greater Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitAO`, bis der Wert des analogen Signals größer als `Value` ist.

`Value`

Datentyp: `num`

Der gewünschte Wert des Signals.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode ERR_WAIT_MAXTIME aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

[\ValueAtTimeout]

Datentyp: num

Bei einem Timeout der Instruktion wird der aktuelle Signalwert in dieser Variablen gespeichert. Die Variable wird nur gesetzt, wenn die Systemvariable ERRNO auf ERR_WAIT_MAXTIME gesetzt ist.

[\Visualize]

Datentyp: switch

Wenn ausgewählt, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der Bedingung, die nicht erfüllt ist, aus Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

[\Header]

Datentyp: string

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein\Header Argument eingegeben wird, wird eine Standardnachricht angezeigt.

[\Message]

Datentyp: string

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

[\MsgArray]

(Message Array)

Datentyp: string

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter \Message oder \MsgArray verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

[\Wrap]

Datentyp: switch

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument \MsgArray zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument \MsgArray in einer eigenen Zeile auf dem Display angezeigt.

[\Icon]

Datentyp: icondata

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs icondata verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.325 WaitAO - Warten, bis ein analoger Ausgangssignalwert gesetzt wird
RobotWare - OS
Fortsetzung

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

`[\VisualizeTime]`

Datentyp: `num`

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente `\VisualizeTime` und `\MaxTime` verwendet werden, muss die im Argument `\MaxTime` verwendete Zeit größer sein als die Zeit, die im Argument `\VisualizeTime` verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument `\VisualizeTime` ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

`[\UIActiveSignal]`

Datentyp: `signaldo`

Das digitale Ausgangssignal, das im optionalen Argument `UIActiveSignal`verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

`[\ErrorNumber]`

Error number

Datentyp: `errnum`

Eine Variable (vor Verwendung vom System auf 0 gesetzt), die den Fehlercode enthält, wenn die Anweisung endet, bevor das Signal den gewünschten Wert hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn diese optionale Variable weggelassen wird, dann wird die Fehlerbehandlung ausgeführt. Die Konstanten ERR_AO_LIM, ERR_NO_ALIASIO_DEF, ERR_NORUNUNIT, ERR_SIG_NOT_VALID und ERR_WAIT_MAXTIME können für die Auswahl der Begründung verwendet werden.

Programmabarbeitung

Wenn der Wert des Signals beim Abarbeiten der Instruktion korrekt ist, fährt das Programm einfach mit der nachfolgenden Instruktion fort.

Wenn der Signalwert nicht korrekt ist, geht der Roboter in einen Wartestatus über, und sobald das Signal den korrekten Wert annimmt, fährt das Programm fort. Die Änderung wird mit einem Interrupt ermittelt, der schnell reagiert (nicht abgefragt wird).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem optionalen Argument \MaxTime angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Wenn die Programmabarbeitung angehalten wurde und später neu gestartet wird, wertet die Instruktion den aktuellen Wert des Signals aus. Jede Änderung während des Programmstopps wird zurückgewiesen.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter \Visualize verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein \Header-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der WaitAO-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AO_LIM	Das programmierte Value-Argument für das angegebene analoge Ausgangssignal Signal liegt außerhalb des zulässigen Bereichs.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.325 WaitAO - Warten, bis ein analoger Ausgangssignalwert gesetzt wird
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
ERR_WAIT_MAXTIME	Es liegt ein Timeout vor (Parameter \MaxTime), bevor das Signal zum richtigen Wert wechselt.

Weitere Beispiele

Weitere Beispiele für die Instruktion WaitAO sind unten aufgeführt.

Beispiel 1:

```
VAR num myvalattimeout:=0;
WaitAO ao1, \LT, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of ao1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Die Programmabarbeitung wird nur fortgesetzt, wenn ao1 kleiner als 5 ist oder wenn ein Timeout erfolgt. Bei einem Timeout kann der Wert des Signals ao1 zum Zeitpunkt des Timeouts ohne ein weiteres Lesen des Signals protokolliert werden.

Beispiel 2:

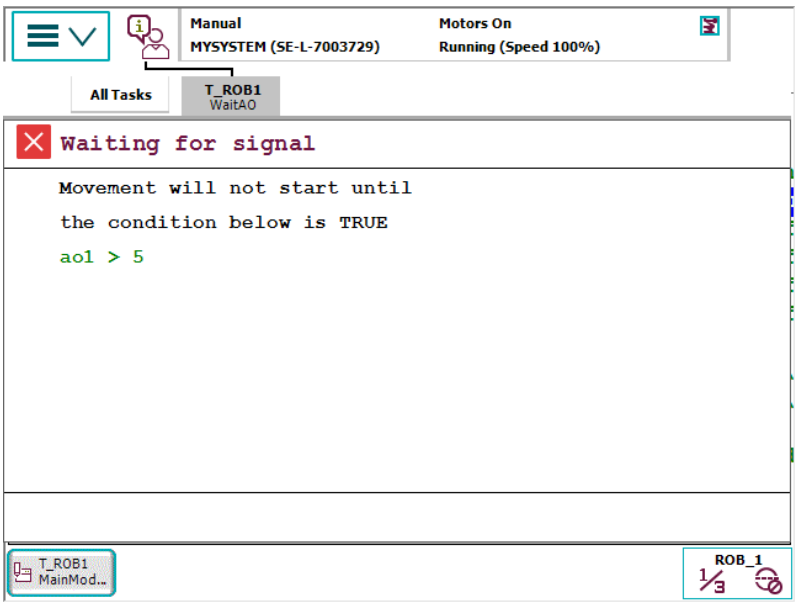
```
WaitAO ao1 \GT, 5 \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn diese Vorgabe nicht erfüllt wird, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen \Header wiedergegeben und \MsgArray wird auf

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

der Anzeige des FlexPendant gemeinsam mit der Bedingung, die nicht erfüllt wurde, angezeigt.



xx1600000151

Syntax

```
WaitAO
  [ Signal ':=' ] <variable (VAR) of signalao> ','
  [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] <expression (IN) of num>
  ['\'MaxTime ':='<expression (IN) of num>]
  ['\'ValueAtTimeout' :=' <variable (VAR) of num>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>]
  ['\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
       ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Voraussetzung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*
Warten, bis eine vorgegebene Dauer verstrichen ist	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*
Warten, bis ein analoger Eingang gesetzt/zurückgesetzt wird	*WaitAI - Warten, bis ein analoger Eingangssignalwert gesetzt wird auf Seite 1040*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.326 WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist

Verwendung

`WaitDI` (*Wait Digital Input*) wird verwendet, um zu warten, bis ein digitaler Eingang gesetzt ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WaitDI`:

Beispiel 1:

```
WaitDI di4, 1;
```

Die Programmabarbeitung fährt erst fort, nachdem der Eingang `di4` gesetzt wurde.

Beispiel 2:

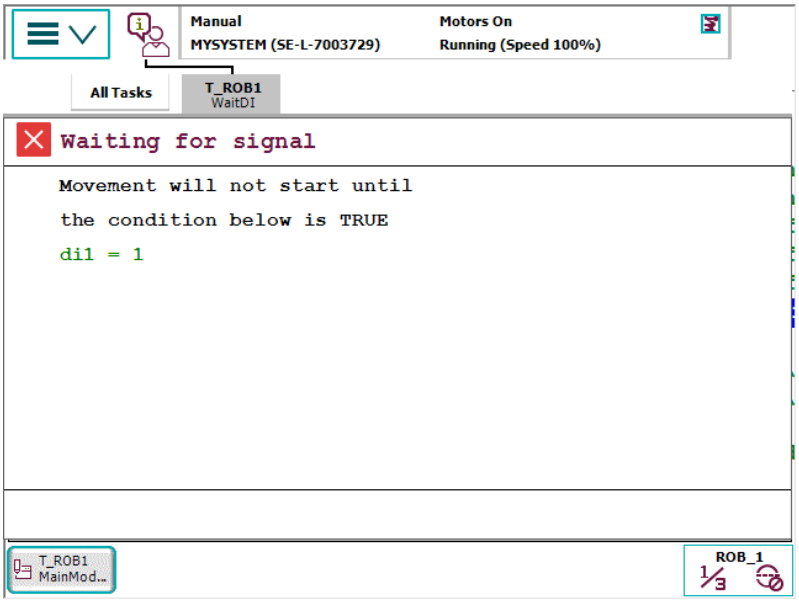
```
WaitDI grip_status, 0;
```

Die Programmabarbeitung fährt erst fort, nachdem der Eingang `grip_status` zurückgesetzt wurde.

Beispiel 3

```
WaitDI di1, 1, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn diese Vorgabe nicht erfüllt wird, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen `\Header` wiedergegeben und `\MsgArray` wird auf der Anzeige des FlexPendant gemeinsam mit der Bedingung, die nicht erfüllt wurde, angezeigt.



xx1600000148

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
WaitDI Signal Value [\MaxTime] [\TimeFlag] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime] [\UIActiveSignal] [\ErrorNumber]
```

`Signal`

Datentyp: `signaldi`

Name des Signals.

`Value`

Datentyp: `dionum`

Der gewünschte Wert des Signals.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, und das Argument `TimeFlag` oder das Argument `ErrorNumber` nicht verwendet wird, kann die Fehlerbehandlung über den Fehlercode `ERR_WAIT_MAXTIME` aufgerufen werden. Wenn keine Fehlerbehandlung vorhanden ist, wird die Abarbeitung gestoppt.

`[\TimeFlag]`

Timeout Flag

Datentyp: `bool`

Der Ausgabeparameter, der den Wert TRUE enthält, wenn die maximal zulässige Wartezeit abläuft, bevor die Bedingung erfüllt wird. Wenn dieser Parameter in der Instruktion angegeben wird, wird das Ablaufen der maximalen Wartezeit nicht als Fehler betrachtet. Dieses Argument wird ignoriert, wenn das Argument `MaxTime` nicht in der Instruktion angegeben wird.

Hinweis

Wenn sowohl `TimeFlag` als auch `ErrorNumber` verwendet werden, dann wird das `TimeFlag` ignoriert. Die boolesche Variable wird bei einem Timeout der Anweisung nicht verwendet.

`[\Visualize]`

Datentyp: `switch`

Wenn ausgewählt, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der Bedingung, die nicht erfüllt ist, aus Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

`[\Header]`

Datentyp: `string`

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein`\Header` Argument eingegeben wird, wird eine Standardnachricht angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.326 WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist
RobotWare - OS
Fortsetzung

`[\Message]`

Datentyp: `string`

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

`[\MsgArray]`

(Message Array)

Datentyp: `string`

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile auf dem Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\VisualizeTime]

Datentyp: num

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente \VisualizeTime und \MaxTime verwendet werden, muss die im Argument \MaxTime verwendete Zeit größer sein als die Zeit, die im Argument \VisualizeTime verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument \VisualizeTime ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignalverwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

[\ErrorNumber]

Error number

Datentyp: errnum

Eine Variable (vor Verwendung vom System auf 0 gesetzt), die den Fehlercode enthält, wenn die Anweisung endet, bevor das Signal den gewünschten Wert hat.

Wenn diese optionale Variable weggelassen wird, dann wird die Fehlerbehandlung ausgeführt . Die Konstanten ERR_NO_ALIASIO_DEF, ERR_NORUNUNIT, ERR_SIG_NOT_VALID und ERR_WAIT_MAXTIME können für die Auswahl der Begründung verwendet werden.

Programmabarbeitung

Wenn der Wert des Signals beim Abarbeiten der Instruktion korrekt ist, fährt das Programm einfach mit der nachfolgenden Instruktion fort.

Wenn der Signalwert nicht korrekt ist, geht der Roboter in einen Wartestatus über, und sobald das Signal den korrekten Wert annimmt, fährt das Programm fort. Die Änderung wird mit einem Interrupt ermittelt, der schnell reagiert (nicht abgefragt wird).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem optionalen Argument \MaxTime angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Wenn die Programmabarbeitung angehalten wurde und später neu gestartet wird, wertet die Instruktion den aktuellen Wert des Signals aus. Jede Änderung während des Programmstopps wird zurückgewiesen.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.326 WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist
RobotWare - OS
Fortsetzung

die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter `\Visualize` verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein `\Header`-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der `WaitDI`-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
`ERR_WAIT_MAXTIME`	Es liegt ein Timeout vor (Parameter `\MaxTime`), bevor das Signal zum richtigen Wert wechselt.

Syntax

```
WaitDI
  [ Signal ':=' ] <variable (VAR) of signaldi>' ,'
  [ Value ':=' ] <expression (IN) of dionum>
  ['\'MaxTime' :='<expression (IN) of num>]
  ['\'TimeFlag':='<variable (VAR) of bool>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>]
  ['\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
       ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Voraussetzung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.326 WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist

RobotWare - OS

Fortsetzung

Information über	Siehe
Warten, bis eine vorgegebene Dauer verstrichen ist	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.327 WaitDO - Warten, bis ein digitales Ausgangssignal gesetzt ist

Verwendung

`WaitDO` (*Wait Digital Output*) wird verwendet, um zu warten, bis ein digitaler Ausgang gesetzt ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WaitDO`:

Beispiel 1:

```
WaitDO do4, 1;
```

Die Programmabarbeitung fährt erst fort, nachdem der Ausgang `do4` gesetzt wurde.

Beispiel 2:

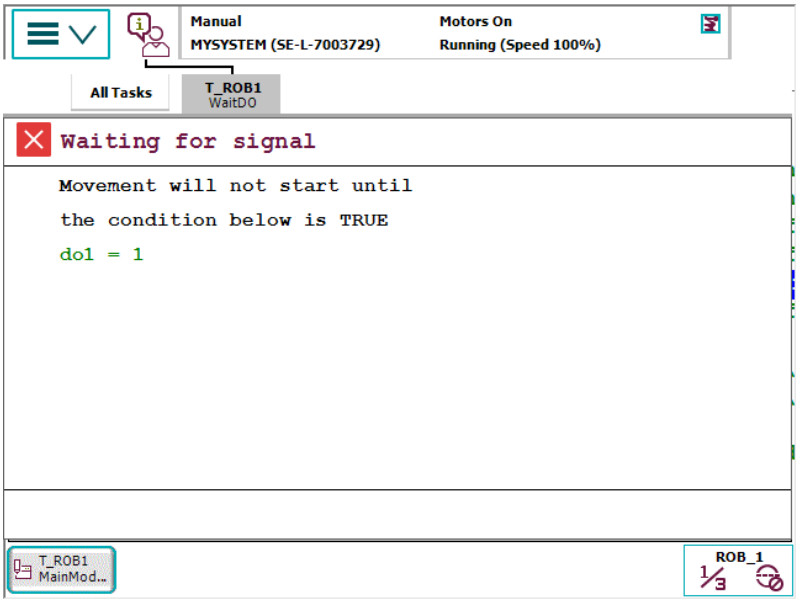
```
WaitDO grip_status, 0;
```

Die Programmabarbeitung fährt erst fort, nachdem der Ausgang `grip_status` zurückgesetzt wurde.

Beispiel 3

```
WaitDO do1, 1, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn diese Vorgabe nicht erfüllt wird, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen `\Header` wiedergegeben und `\MsgArray` wird auf der Anzeige des FlexPendant gemeinsam mit der Bedingung, die nicht erfüllt wurde, angezeigt.



xx1600000149

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
WaitDO Signal Value [\MaxTime] [\TimeFlag] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime] [\UIActiveSignal] [\ErrorNumber]
```

`Signal`

Datentyp: `signaldo`

Name des Signals.

`Value`

Datentyp: `dionum`

Der gewünschte Wert des Signals.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, und das Argument `TimeFlag` oder das Argument `ErrorNumber` nicht verwendet wird, kann die Fehlerbehandlung über den Fehlercode `ERR_WAIT_MAXTIME` aufgerufen werden. Wenn keine Fehlerbehandlung vorhanden ist, wird die Abarbeitung gestoppt.

`[\TimeFlag]`

Timeout Flag

Datentyp: `bool`

Der Ausgabeparameter, der den Wert TRUE enthält, wenn die maximal zulässige Wartezeit abläuft, bevor die Bedingung erfüllt wird. Wenn dieser Parameter in der Instruktion angegeben wird, wird das Ablaufen der maximalen Wartezeit nicht als Fehler betrachtet. Dieses Argument wird ignoriert, wenn das Argument `MaxTime` nicht in der Instruktion angegeben wird.

Hinweis

Wenn sowohl `TimeFlag` als auch `ErrorNumber` verwendet werden, dann wird das `TimeFlag` ignoriert. Die boolesche Variable wird bei einem Timeout der Anweisung nicht verwendet.

`[\Visualize]`

Datentyp: `switch`

Wenn ausgewählt, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der Bedingung, die nicht erfüllt ist, aus Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

`[\Header]`

Datentyp: `string`

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein`\Header` Argument eingegeben wird, wird eine Standardnachricht angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.327 WaitDO - Warten, bis ein digitales Ausgangssignal gesetzt ist
RobotWare - OS
Fortsetzung

`[\Message]`

Datentyp: `string`

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

`[\MsgArray]`

(Message Array)

Datentyp: `string`

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile auf dem Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\VisualizeTime]

Datentyp: num

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente \VisualizeTime und \MaxTime verwendet werden, muss die im Argument \MaxTime verwendete Zeit größer sein als die Zeit, die im Argument \VisualizeTime verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument \VisualizeTime ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignalverwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

[\ErrorNumber]

Error number

Datentyp: errnum

Eine Variable (vor Verwendung vom System auf 0 gesetzt), die den Fehlercode enthält, wenn die Anweisung endet, bevor das Signal den gewünschten Wert hat.

Wenn diese optionale Variable weggelassen wird, dann wird die Fehlerbehandlung ausgeführt . Die Konstanten ERR_NO_ALIASIO_DEF, ERR_NORUNUNIT, ERR_SIG_NOT_VALID und ERR_WAIT_MAXTIME können für die Auswahl der Begründung verwendet werden.

Programmabarbeitung

Wenn der Wert des Ausgangssignals beim Abarbeiten der Instruktion korrekt ist, fährt das Programm einfach mit der nachfolgenden Instruktion fort.

Wenn der Wert des Ausgangssignals nicht korrekt ist, geht der Roboter in einen Wartestatus über. Sobald das Signal den korrekten Wert annimmt, wird das Programm fortgesetzt. Die Änderung wird mit einem Interrupt ermittelt, der schnell reagiert (nicht abgefragt wird).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem optionalen Argument \MaxTime angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Wenn die Programmabarbeitung angehalten wurde und später neu gestartet wird, wertet die Instruktion den aktuellen Wert des Signals aus. Jede Änderung während des Programmstopps wird zurückgewiesen.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.327 WaitDO - Warten, bis ein digitales Ausgangssignal gesetzt ist
RobotWare - OS
Fortsetzung

die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter `\Visualize` verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein `\Header`-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der `WaitDO`-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
`ERR_WAIT_MAXTIME`	Es liegt ein Timeout vor (Parameter `\MaxTime`), bevor das Signal zum richtigen Wert wechselt.

Syntax

```
WaitDO
  [ Signal ':='] <variable (VAR) of signaldo>','
  [ Value ':=' ] <expression (IN) of dionum>
  ['\'MaxTime' :='<expression (IN) of num>]
  ['\'TimeFlag':='<variable (VAR) of bool>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' VisualizeTime ':=' <expression (IN) of num>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>]
  ['\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
       ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Voraussetzung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Warten, bis eine vorgegebene Dauer verstrichen ist	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*
Warten, bis ein Eingang gesetzt/zurückgesetzt wird	*WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist auf Seite 1052*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.328 WaitGI - Warten, bis eine Gruppe digitaler Eingangssignale gesetzt wird

Verwendung

`WaitGI` (*Wait Group digital Input*) wird verwendet, um zu warten, bis eine Gruppe digitaler Eingangssignale auf angegebene Werte gesetzt wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitGI`:

Siehe auch *Weitere Beispiele auf Seite 1068*.

Beispiel 1:

```
WaitGI gi4, 5;
```

Die Programmabarbeitung fährt erst fort, nachdem der Eingang `gi4` den Wert 5 aufweist.

Beispiel 2:

```
WaitGI grip_status, 0;
```

Die Programmabarbeitung fährt erst fort, nachdem der Eingang `grip_status` zurückgesetzt wurde.

Argumente

```
WaitGI Signal [\NOTEQ] | [\LT] | [\GT] Value | Dvalue [\MaxTime]
     [\ValueAtTimeout] | [\DvalueAtTimeout] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime] [\UIActiveSignal] [\ErrorNumber]
```

`Signal`

Datentyp: `signalgi`

Der Name der Gruppe digitaler Eingangssignale.

`[\NOTEQ]`

NOT EQual

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitGI`, bis der Wert des digitalen Gruppensignals von `Value` abweicht.

`[\LT]`

Less Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitGI`, bis der Wert des digitalen Gruppensignals kleiner als `Value` ist.

`[\GT]`

Greater Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitGI`, bis der Wert des digitalen Gruppensignals größer als `Value` ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Value`

Datentyp: `num`

Der gewünschte Wert des Signals. Muss eine Ganzzahl im Gültigkeitsbereich des verwendeten digitalen Gruppeneingangssignals sein. Der zulässige Wert hängt von der Anzahl der Signale in der Gruppe ab. Der Höchstwert beim `Value`-Argument ist 8388608, und das entspricht dem Höchstwert für ein digitales Signal mit 32 Bit.

`Dvalue`

Datentyp: `dnum`

Der gewünschte Wert des Signals. Muss eine Ganzzahl im Gültigkeitsbereich des verwendeten digitalen Gruppeneingangssignals sein. Der zulässige Wert hängt von der Anzahl der Signale in der Gruppe ab. Die Höchstanzahl an Bits für ein digitales Gruppensignal beträgt 32 Bits. Mit einer `dnum`-Variablen kann der Wertebereich 0-4294967295 abgedeckt werden, der dem Wertebereich eines digitalen Signals mit 32 Bits entspricht.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_WAIT_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

`[\ValueAtTimeout]`

Datentyp: `num`

Bei einem Timeout der Instruktion wird der aktuelle Signalwert in dieser Variablen gespeichert. Die Variable wird nur gesetzt, wenn die Systemvariable `ERRNO` auf `ERR_WAIT_MAXTIME` gesetzt ist. Wenn das Argument `Dvalue` verwendet wird, speichern Sie den aktuellen Wert des Signals mithilfe von `DvalueAtTimeout` (Grund: Begrenzung des maximalen ganzzahligen Wertes für `num`).

Signalwerte zwischen 0 und 8388608 werden immer exakte Ganzzahlen gespeichert.

`[\DvalueAtTimeout]`

Datentyp: `dnum`

Bei einem Timeout der Instruktion wird der aktuelle Signalwert in dieser Variablen gespeichert. Die Variable wird nur gesetzt, wenn die Systemvariable `ERRNO` auf `ERR_WAIT_MAXTIME` gesetzt ist.

Signalwerte zwischen 0 und 4294967295 werden immer exakte Ganzzahlen gespeichert.

`[\Visualize]`

Datentyp: `switch`

Wenn ausgewählt, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der Bedingung, die nicht erfüllt ist, aus Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.328 WaitGI - Warten, bis eine Gruppe digitaler Eingangssignale gesetzt wird
RobotWare - OS
Fortsetzung

`[\Header]`

Datentyp: `string`

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein`\Header` Argument eingegeben wird, wird eine Standardnachricht angezeigt.

`[\Message]`

Datentyp: `string`

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

`[\MsgArray]`

(Message Array)

Datentyp: `string`

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile auf dem Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

`[\VisualizeTime]`

Datentyp: `num`

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente `\VisualizeTime` und `\MaxTime` verwendet werden, muss die im Argument `\MaxTime` verwendete Zeit größer sein als die Zeit, die im Argument `\VisualizeTime` verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument `\VisualizeTime` ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

`[\UIActiveSignal]`

Datentyp: `signaldo`

Das digitale Ausgangssignal, das im optionalen Argument `UIActiveSignal`verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

`[\ErrorNumber]`

Error number

Datentyp: `errnum`

Eine Variable (vor Verwendung vom System auf 0 gesetzt), die den Fehlercode enthält, wenn die Anweisung endet, bevor das Signal den gewünschten Wert hat.

Wenn diese optionale Variable weggelassen wird, dann wird die Fehlerbehandlung ausgeführt. Die Konstanten `ERR_GO_LIM`, `ERR_NO_ALIASIO_DEF`, `ERR_NORUNUNIT`, `ERR_SIG_NOT_VALID` und `ERR_WAIT_MAXTIME` können für die Auswahl der Begründung verwendet werden.

Programmabarbeitung

Wenn der Wert des Signals beim Abarbeiten der Instruktion korrekt ist, fährt das Programm einfach mit der nachfolgenden Instruktion fort.

Wenn der Signalwert falsch ist, geht der Roboter in einen Wartestatus über, und sobald das Signal den korrekten Wert annimmt, fährt das Programm fort. Die Änderung wird mit einem Interrupt ermittelt, der schnell reagiert (nicht abgefragt wird).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem optionalen Argument `\MaxTime` angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.328 WaitGI - Warten, bis eine Gruppe digitaler Eingangssignale gesetzt wird
RobotWare - OS
Fortsetzung

Wenn die Programmabarbeitung angehalten wurde und später neu gestartet wird, wertet die Instruktion den aktuellen Wert des Signals aus. Jede Änderung während des Programmstopps wird zurückgewiesen.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter `\Visualize` verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein `\Header`-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der `WaitGI`-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_GO_LIM`	Das programmierte `Value`- oder `Dvalue`-Argument für das angegebene digitale Gruppeneingangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
`ERR_WAIT_MAXTIME`	Es liegt ein Timeout vor (Parameter `\MaxTime`), bevor das Signal zum richtigen Wert wechselt.

Weitere Beispiele

Weitere Beispiele für die Instruktion `WaitGI` sind unten aufgeführt.

Beispiel 1:

```
WaitGI gi1,\NOTEQ,0;
```

Die Programmabarbeitung fährt erst fort, nachdem `gi1` vom Wert 0 abweicht.

Beispiel 2:

```
WaitGI gi1,\LT,1;
```

Die Programmabarbeitung fährt erst fort, nachdem `gi1` kleiner als 1 ist.

Beispiel 3

```
WaitGI gi1,\GT,0;
```

Die Programmabarbeitung fährt erst fort, nachdem `gi1` größer als 0 ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 4:

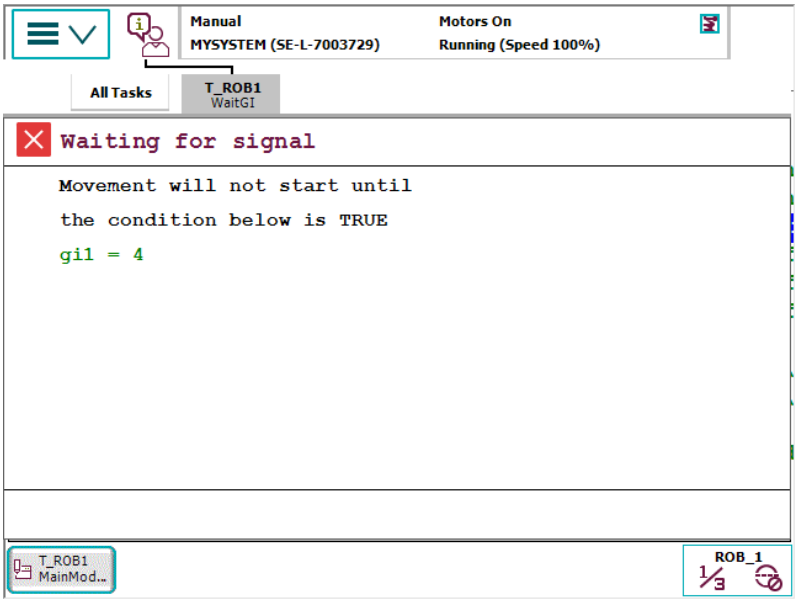
```
VAR num myvalattimeout:=0;
WaitGI gi1, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of gi1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Die Programmabarbeitung wird nur fortgesetzt, wenn `gi1` gleich 5 ist oder wenn ein Timeout eintritt. Bei einem Timeout kann der Wert des Signals `gi1` zum Zeitpunkt des Timeouts ohne ein weiteres Lesen des Signals protokolliert werden.

Beispiel 5:

```
WaitGI gi1, 4, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn diese Vorgabe nicht erfüllt wird, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen `\Header` wiedergegeben und `\MsgArray` wird auf der Anzeige des FlexPendant gemeinsam mit der Bedingung, die nicht erfüllt wurde, angezeigt.



xx1600000152

Syntax

```
WaitGI
  [ Signal ':=' ] <variable (VAR) of signalgi> ','
  ['\' NOTEQ] | [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] <expression (IN) of num>
  | [ Dvalue' :=' ] <expression (IN) of dnum>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.328 WaitGI - Warten, bis eine Gruppe digitaler Eingangssignale gesetzt wird

RobotWare - OS

Fortsetzung

```
['\'MaxTime ':='<expression (IN) of num>]
['\'ValueAtTimeout' :=' <variable (VAR) of num> ]
| [ '\'DvalueAtTimeout' :=' <variable (VAR) of dnum> ]
['\' Visualize]
['\' Header ':=' <expression (IN) of string>]]
['\' Message ':=' <expression (IN) of string>]
| ['\' MsgArray ':=' <array {*} (IN) of string>]
['\' Wrap]
['\' Icon ':=' <expression (IN) of icondata>]
['\' Image ':=' <expression (IN) of string>]
['\' VisualizeTime ':=' <expression (IN) of num>]
['\' UIActiveSignal ':=' <variable (VAR) of signaldo>]
['\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
     ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Voraussetzung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*
Warten, bis eine vorgegebene Dauer verstrichen ist	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*
Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt/zurückgesetzt wird	*WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird auf Seite 1071*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.329 WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird

Verwendung

Mit `WaitGO` (*Wait Group digital Output*) kann gewartet werden, bis eine Gruppe digitaler Ausgangssignale auf einen bestimmten Wert gesetzt wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WaitGO`:

Siehe auch *Weitere Beispiele auf Seite 1075*.

Beispiel 1:

```
WaitGO go4, 5;
```

Die Programmabarbeitung fährt erst fort, nachdem der Ausgang `go4` den Wert 5 aufweist.

Beispiel 2:

```
WaitGO grip_status, 0;
```

Die Programmabarbeitung fährt erst fort, nachdem der Ausgang `grip_status` zurückgesetzt wurde.

Argumente

```
WaitGO Signal [\NOTEQ] | [\LT] | [\GT] Value | Dvalue [\MaxTime]
     [\ValueAtTimeout] | [\DvalueAtTimeout] [\Visualize] [\Header]
     [\Message] | [\MsgArray] [\Wrap] [\Icon] [\Image]
     [\VisualizeTime] [\UIActiveSignal] [\ErrorNumber]
```

`Signal`

Datentyp: `signalgo`

Der Name der Gruppe digitaler Ausgangssignale.

`[\NOTEQ]`

NOT EQual

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitGO`, bis der Wert des digitalen Gruppensignals von `Value` abweicht.

`[\LT]`

Less Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitGO`, bis der Wert des digitalen Gruppensignals kleiner als `Value` ist.

`[\GT]`

Greater Than

Datentyp: `switch`

Bei Verwendung dieses Parameters wartet die Instruktion `WaitGO`, bis der Wert des digitalen Gruppensignals größer als `Value` ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.329 WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird
RobotWare - OS
Fortsetzung

`Value`

Datentyp: `num`

Der gewünschte Wert des Signals. Muss eine Ganzzahl im Gültigkeitsbereich des verwendeten digitalen Gruppenausgangssignals sein. Der zulässige Wert hängt von der Anzahl der Signale in der Gruppe ab. Der Höchstwert beim `Value`-Argument ist 8388608, und das entspricht dem Höchstwert für ein digitales Signal mit 32 Bit.

`Dvalue`

Datentyp: `dnum`

Der gewünschte Wert des Signals. Muss eine Ganzzahl im Gültigkeitsbereich des verwendeten digitalen Gruppenausgangssignals sein. Der zulässige Wert hängt von der Anzahl der Signale in der Gruppe ab. Die Höchstanzahl an Bits für ein digitales Gruppensignal beträgt 32 Bits. Mit einer `dnum`-Variablen kann der Wertebereich 0-4294967295 abgedeckt werden, der dem Wertebereich eines digitalen Signals mit 32 Bits entspricht.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_WAIT_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

`[\ValueAtTimeout]`

Datentyp: `num`

Bei einem Timeout der Instruktion wird der aktuelle Signalwert in dieser Variablen gespeichert. Die Variable wird nur gesetzt, wenn die Systemvariable `ERRNO` auf `ERR_WAIT_MAXTIME` gesetzt ist. Wenn das Argument `Dvalue` verwendet wird, speichern Sie den aktuellen Wert des Signals mithilfe von `DvalueAtTimeout` (Grund: Begrenzung des maximalen ganzzahligen Wertes für `num`).

Signalwerte zwischen 0 und 8388608 werden immer exakte Ganzzahlen gespeichert.

`[\DvalueAtTimeout]`

Datentyp: `dnum`

Bei einem Timeout der Instruktion wird der aktuelle Signalwert in dieser Variablen gespeichert. Die Variable wird nur gesetzt, wenn die Systemvariable `ERRNO` auf `ERR_WAIT_MAXTIME` gesetzt ist.

Signalwerte zwischen 0 und 4294967295 werden immer exakte Ganzzahlen gespeichert.

`[\Visualize]`

Datentyp: `switch`

Wenn ausgewählt, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der Bedingung, die nicht erfüllt ist, aus Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\Header]

Datentyp: string

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein\Header Argument eingegeben wird, wird eine Standardnachricht angezeigt.

[\Message]

Datentyp: string

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

[\MsgArray]

(Message Array)

Datentyp: string

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter \Message oder \MsgArray verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

[\Wrap]

Datentyp: switch

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument \MsgArray zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument \MsgArray in einer eigenen Zeile auf dem Display angezeigt.

[\Icon]

Datentyp: icondata

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs icondata verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Standardmäßig wird kein Icon angezeigt.

[\Image]

Datentyp: string

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.329 WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird
RobotWare - OS
Fortsetzung

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

[\VisualizeTime]

Datentyp: num

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente \VisualizeTime und \MaxTime verwendet werden, muss die im Argument \MaxTime verwendete Zeit größer sein als die Zeit, die im Argument \VisualizeTime verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument \VisualizeTime ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignalverwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

[\ErrorNumber]

Error number

Datentyp: errnum

Eine Variable (vor Verwendung vom System auf 0 gesetzt), die den Fehlercode enthält, wenn die Anweisung endet, bevor das Signal den gewünschten Wert hat.

Wenn diese optionale Variable weggelassen wird, dann wird die Fehlerbehandlung ausgeführt. Die Konstanten ERR_GO_LIM, ERR_NO_ALIASIO_DEF, ERR_NORUNUNIT, ERR_SIG_NOT_VALID und ERR_WAIT_MAXTIME können für die Auswahl der Begründung verwendet werden.

Programmabarbeitung

Wenn der Wert des Signals beim Abarbeiten der Instruktion korrekt ist, fährt das Programm einfach mit der nachfolgenden Instruktion fort.

Wenn der Signalwert nicht korrekt ist, geht der Roboter in einen Wartestatus über, und sobald das Signal den korrekten Wert annimmt, fährt das Programm fort. Die Änderung wird mit einem Interrupt ermittelt, der schnell reagiert (nicht abgefragt wird).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem optionalen Argument \MaxTime angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn die Programmabarbeitung angehalten wurde und später neu gestartet wird, wertet die Instruktion den aktuellen Wert des Signals aus. Jede Änderung während des Programmstopps wird zurückgewiesen.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter `\Visualize` verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein `\Header`-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der `WaitGO`-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_GO_LIM`	Das programmierte `Value`- oder `Dvalue`-Argument für das angegebene digitale Gruppenausgangssignal `Signal` liegt außerhalb des zulässigen Bereichs.
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
`ERR_WAIT_MAXTIME`	Es liegt ein Timeout vor (Parameter `\MaxTime`), bevor das Signal zum richtigen Wert wechselt.

Weitere Beispiele

Weitere Beispiele für die Instruktion `WaitGO` sind unten aufgeführt.

Beispiel 1:

```
WaitGO go1,\NOTEQ,0;
```

Die Programmabarbeitung fährt erst fort, nachdem `go1` vom Wert 0 abweicht.

Beispiel 2:

```
WaitGO go1,\LT,1;
```

Die Programmabarbeitung fährt erst fort, nachdem `go1` kleiner als 1 ist.

Beispiel 3

```
WaitGO go1,\GT,0;
```

Die Programmabarbeitung fährt erst fort, nachdem `go1` größer als 0 ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.329 WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird

RobotWare - OS

Fortsetzung

Beispiel 4:

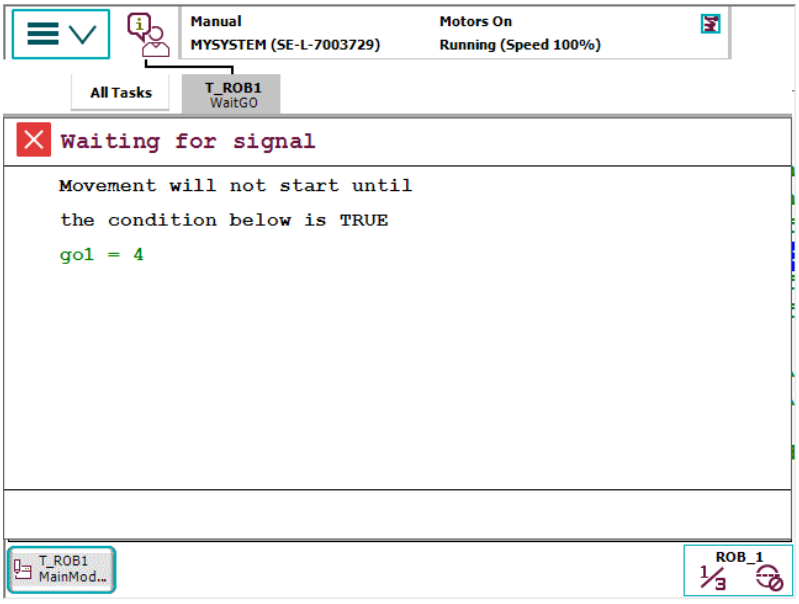
```
VAR num myvalattimeout:=0;
WaitGO go1, 5 \MaxTime:=4 \ValueAtTimeout:=myvalattimeout;
ERROR
  IF ERRNO=ERR_WAIT_MAXTIME THEN
    TPWrite "Value of go1 at timeout:" + ValToStr(myvalattimeout);
    TRYNEXT;
  ELSE
    ! No error recovery handling
  ENDIF
```

Die Programmabarbeitung wird nur fortgesetzt, wenn `go1` gleich 5 ist oder wenn ein Timeout eintritt. Bei einem Timeout kann der Wert des Signals `go1` zum Zeitpunkt des Timeouts ohne ein weiteres Lesen des Signals protokolliert werden.

Beispiel 5:

```
WaitGO go1, 4, \Visualize \Header:="Waiting for signal"
     \MsgArray:=["Movement will not start until", "the condition
     below is TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn diese Vorgabe nicht erfüllt wird, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen `\Header` wiedergegeben und `\MsgArray` wird auf der Anzeige des FlexPendant gemeinsam mit der Bedingung, die nicht erfüllt wurde, angezeigt.



xx1600000153

Syntax

```
WaitGO
  [ Signal ':=' ] <variable (VAR) of signalgo> ','
  ['\' NOTEQ] | [ '\' LT] | [ '\' GT] ','
  [ Value ':=' ] <expression (IN) of num>
  | [ Dvalue ':=' ] <expression (IN) of dnum>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.329 WaitGO – Warten, bis eine Gruppe digitaler Ausgangssignale gesetzt wird

RobotWare - OS

Fortsetzung

```
['\' MaxTime ':=' <expression (IN) of num>]
['\' ValueAtTimeout ':=' <variable (VAR) of num> ]
| [ '\' DvalueAtTimeout ':=' <variable (VAR) of dnum> ]
['\' Visualize]
['\' Header ':=' <expression (IN) of string>]]
['\' Message ':=' <expression (IN) of string>]
| ['\' MsgArray ':=' <array {*} (IN) of string>]
['\' Wrap]
['\' Icon ':=' <expression (IN) of icondata>]
['\' Image ':=' <expression (IN) of string>]
['\' VisualizeTime ':=' <expression (IN) of num>]
['\' UIActiveSignal ':=' <variable (VAR) of signaldo>]
['\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
    ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Voraussetzung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*
Warten, bis eine vorgegebene Dauer verstrichen ist	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*
Warten, bis eine Gruppe digitaler Eingangssignale gesetzt/zurückgesetzt wird	*WaitGI - Warten, bis eine Gruppe digitaler Eingangssignale gesetzt wird auf Seite 1064*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.330 WaitLoad – Das geladene Modul mit der Task verbinden

Verwendung

`WaitLoad` wird verwendet, um das mit der `StartLoad`-Instruktion geladene Modul mit der Programmtask zu verbinden.

Das geladene Programmmodul wird den bereits bestehenden Modulen im Programmspeicher hinzugefügt.

Ein mit `StartLoad` geladenes Modul muss mit der Programmtask mit der `WaitLoad`-Instruktion verbunden werden, bevor seine Symbole oder Routinen verwendet werden können.

`WaitLoad` kann auch ein Programmmodul entladen, wenn die optionalen Schalter verwendet werden. Auf diese Weise wird die Anzahl der Verbindungen minimiert (1 statt 2).

`WaitLoad` kann auch eine Überprüfung auf nicht aufgelöste Verweise durchführen, wenn der optionale Schalter `\CheckRef` verwendet wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitLoad`:

Siehe auch *Weitere Beispiele auf Seite 1080*.

Beispiel 1:

```
VAR loadsession load1;
...
StartLoad "HOME:/PART_A.MOD", load1;
MoveL p10, v1000, z50, tool1 \WObj:=wobj1;
MoveL p20, v1000, z50, tool1 \WObj:=wobj1;
MoveL p30, v1000, z50, tool1 \WObj:=wobj1;
MoveL p40, v1000, z50, tool1 \WObj:=wobj1;
WaitLoad load1;
%"routine_x"%;
UnLoad "HOME:/PART_A.MOD";
```

Lädt das Programmmodul `PART_A.MOD` aus `HOME:` in den Programmspeicher. Parallel dazu wird der Roboter bewegt. Dann wird das neue Programmmodul mit der Programmtask verbunden und die Routine `routine_x` im Modul `PART_A` aufgerufen.

Argumente

```
WaitLoad [\UnloadPath] [\UnloadFile] LoadNo [\CheckRef]
```

`[\UnloadPath]`

Datentyp: `string`

Pfad und Name der Datei, die aus dem Programmspeicher entfernt wird. Der Dateiname soll ausgeschlossen werden, wenn das Argument `\UnloadFile` verwendet wird.

`[\UnloadFile]`

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn der Dateiname im Argument `\UnloadPath` ausgelassen wird, muss er mit diesem Argument definiert werden.

`LoadNo`

Datentyp: `loadsession`

Dies ist ein Verweis auf den Ladevorgang, der durch die Instruktion `StartLoad` erstellt wird, um das geladene Programmmodul mit der Programmtask zu verbinden.

`[\CheckRef]`

Datentyp: `switch`

Überprüfung auf nicht aufgelöste Verweise in der Programmtask nach dem Laden des Moduls. Wenn der Parameter nicht verwendet wird, erfolgt keine Überprüfung auf nicht aufgelöste Verweise.

Programmabarbeitung

Die Instruktion `WaitLoad` wartet zuerst auf das Ende des Ladevorgangs (sofern noch nicht beendet), bevor das Modul verbunden und initialisiert wird. Die Initialisierung des geladenen Moduls setzt alle Variablen auf Modulebene auf ihre Anfangswerte.

Nicht aufgelöste Verweise werden für die Ladevorgänge `StartLoad - WaitLoad` immer akzeptiert, doch bei der Abarbeitung eines nicht aufgelösten Verweises tritt ein Laufzeitfehler auf, wenn der Parameter `\CheckRef` nicht verwendet wird.

Das System startet ggf. den Entladevorgang. Wenn das Entladen des Moduls fehlschlägt, wird kein neues Modul geladen.

Wenn beim Ladevorgang ein Fehler, einschließlich nicht aufgelöster Verweise bei Verwendung des Schalters `\CheckRef`, auftritt, ist das geladene Modul nicht mehr im Programmspeicher verfügbar.

Für eine gute Programmstruktur, die sich leicht verstehen und pflegen lässt, sollten sämtliche Lade- und Entladevorgänge von Programmmodulen im Hauptmodul erfolgen, das stets bei der Abarbeitung im Programmspeicher vorhanden ist.

Informationen zum Laden eines Programms, das eine main-Prozedur für ein Hauptprogramm (mit einer weiteren `main`-Prozedur) enthält, finden Sie unter der Instruktion `Load`.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILNOTFND`	Die in der Instruktion `StartLoad` angegebene Datei wurde nicht gefunden.
`ERR_IOERROR`	Es gibt ein Problem beim Lesen der zu ladenden Datei.
`ERR_UNKPROC`	Das Argument `LoadNo` verweist auf einen unbekannten Ladevorgang.
`ERR_PRGMEMFULL`	Das Modul kann nicht geladen werden, weil der Programmspeicher voll ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.330 WaitLoad – Das geladene Modul mit der Task verbinden
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_LOADED	Das Programmmodul ist bereits in den Programmspeicher geladen.
ERR_SYNTAX	Das geladene Modul enthält Syntaxfehler.
ERR_LINKREF	Das geladene Modul verursacht schwerwiegende Verknüpfungsfehler. • Das geladene Modul verursacht schwerwiegende Verknüpfungsfehler. • WaitLoad wird mit dem Schalter \CheckRef verwendet, um eine Überprüfung auf Verweisfehler durchzuführen, und der Programmspeicher enthält nicht aufgelöste Verweise.

Die folgenden Fehler können nur auftreten, wenn das Argument \UnloadPath in der Instruktion WaitLoad verwendet wird:

Name	Fehlerursache
ERR_UNLOAD	• Das im Argument \UnloadPath angegebene Programmmodul kann wegen einer laufenden Abarbeitung innerhalb des Moduls nicht entladen werden. • Das im Argument \UnloadPath angegebene Programmmodul kann nicht entladen werden, weil es nicht mit Load oder StartLoad-WaitLoad aus dem RAPID-Programm geladen wurde.

Beim Auftreten von einigen dieser Fehler wird das aktuelle Modul entladen und ist in der ERROR-Behandlungsroutine nicht verfügbar.

Weitere Beispiele

Weitere Beispiele für die Instruktion WaitLoad sind unten aufgeführt.

Beispiel 1:

```
StartLoad "HOME:/DOORDIR/DOOR2.MOD", load1;
...
WaitLoad \UnloadPath:="HOME:/DOORDIR/DOOR1.MOD", load1;
```

Lädt das Programmmodul DOOR2.MOD aus HOME: im Verzeichnis DOORDIR in den Programmspeicher und verbindet das neue Modul mit der Task. Das Programmmodul DOOR1.MOD wird aus dem Programmspeicher entfernt.

Beispiel 2:

```
StartLoad "HOME:" \File:="DOORDIR/DOOR2.MOD", load1;
! The robot can do some other work
WaitLoad \UnloadPath:="HOME:" \File:= "DOORDIR/DOOR1.MOD", load1;
```

Dies ist identisch mit den folgenden Instruktionen, jedoch kann der Roboter beim Ladevorgang andere Aktionen ausführen und dies zusätzlich schneller (nur eine Verbindung statt zwei Verbindungen unten).

```
Load "HOME:" \File:="DOORDIR/DOOR2.MOD";
UnLoad "HOME:" \File:="DOORDIR/DOOR1.MOD";
```

Hinweis

RETRY kann nicht für die Fehlerbehandlung von Fehlern aus WaitLoad verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Es ist nicht möglich, den aktuellen Wert einer Variable PERS zu ändern, indem dasselbe Modul mit einem neuen Anfangswert für die aktuelle Variable PERS erneut geladen wird.

Beispiel:

- Die Datei `my_module.mod` mit der Deklaration `PERS num my_pers:=1;` wird in das System geladen.
- Die Datei `my_module.mod` wird auf dem Datenträger mit dem neuen Persistentenwert, z. B. `PERS num my_pers:=3;`, bearbeitet.
- Der nachfolgend angegebene Code wird abgearbeitet.
- Nach dem erneuten Laden von `my_module.mod` ist der Wert von `my_pers` immer noch `1` und nicht `3`.

```
StartLoad \Dynamic, "HOME:/my_module.mod", load1;
...
WaitLoad \UnLoadPath:="HOME:/my_module.mod", load1;
```

Diese Einschränkung ist eine Folge der Eigenschaften von PERS-Variablen. Der aktuelle Wert der PERS-Variablen wird durch den neu geladenen PERS-Anfangswert nicht geändert, wenn die PERS-Variable während des Ladens verwendet wird.

Die oben beschriebenen Probleme treten nicht auf, wenn stattdessen der folgende Code abgearbeitet wird:

```
UnLoad "HOME:/my_module.mod";
StartLoad \Dynamic, "HOME:/my_module.mod", load1;
...
WaitLoad load1;
```

Eine weitere Möglichkeit ist die Verwendung eines CONST als Anfangswert und folgender Zuweisung zu Beginn der Abarbeitung im neuen Modul: `my_pers := my_const;`

Syntax

WaitLoad

```
[ '\' UnloadPath ':=' <expression (IN) of string> ',']
[ '\' UnloadFile ':=' <expression (IN) of string> ',']
[ LoadNo ':=' ] <variable (VAR) of loadsession>
[ '\' CheckRef ] ';'
```

Weitere Informationen

Information über	Siehe
Laden eines Programmmoduls während der Abarbeitung	*StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802*
Ladevorgang	*loadsession – Ladevorgang für ein Programm auf Seite 1730*
Laden eines Programmmoduls	*Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346*
Entfernen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.330 WaitLoad – Das geladene Modul mit der Task verbinden

RobotWare - OS

Fortsetzung

Information über	Siehe
Abbruch des Ladens eines Programmmoduls	*CancelLoad - Abbruch des Ladens eines Moduls auf Seite 81*
Überprüft Programmreferenzen.	*CheckProgRef - Überprüfen von Programmverweisen auf Seite 131*
Prozeduraufruf mit später Bindung	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.331 WaitRob – Bis zum Stopppunkt oder Stillstand warten

Verwendung

`WaitRob` (*Wait Robot*) wartet, bis der Roboter und die externen Achsen den Stopppunkt erreicht haben oder sich nicht mehr bewegen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitRob`:

Siehe auch *Weitere Beispiele auf Seite 1083*.

Beispiel 1:

```
WaitRob \InPos;
```

Die Programmabarbeitung wartet, bis der Roboter und die externen Achsen den Stopppunkt erreicht haben.

Argumente

```
WaitRob [\InPos] | [\ZeroSpeed]
```

`[\InPos]`

In Position

Datentyp: `switch`

Wenn dieses Argument verwendet wird, müssen der Roboter und die externen Achsen den Stopppunkt (`ToPoint` der aktuellen Bewegungsinstruktion) erreicht haben, bevor die Abarbeitung fortgesetzt werden kann.

`[\ZeroSpeed]`

Zero Speed

Datentyp: `switch`

Wenn dieses Argument verwendet wird, müssen der Roboter und die externen Achsen die Geschwindigkeit null aufweisen, bevor die Abarbeitung fortgesetzt werden kann.

Wenn keines der Argumente `\InPos` oder `\ZeroSpeed` eingegeben wurde, wird eine Fehlermeldung angezeigt.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `WaitRob` sind unten aufgeführt.

Beispiel 1:

```
PROC stop_event()
  WaitRob \ZeroSpeed;
  SetDO rob_moving, 0;
ENDPROC
```

Das Beispiel zeigt eine Ereignisroutine, die bei einem Programmstopp ausgeführt wird. Das digitale Ausgangssignal `rob_moving` ist 1, solange sich der Roboter bewegt, und wird auf 0 gesetzt, wenn der Roboter und die externen Achsen nach einem Programmstopp angehalten wurden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.331 WaitRob – Bis zum Stopppunkt oder Stillstand warten

RobotWare - OS

Fortsetzung

Syntax

```
WaitRob
  [ '\' InPos ] | [ '\' ZeroSpeed ] ';'
```

Weitere Informationen

Information über	Siehe
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles*
Andere Positionierungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*
Definition von Stopppunktdaten	*stoppointdata - Stopppunktdaten auf Seite 1799*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.332 WaitSensor - Auf die Sensorverbindung warten

Verwendung

WaitSensor verbindet mit einem Objekt im Startfenster an einer sensormechanischen Einheit.

Grundlegende Beispiele

Grundlegende Beispiele für die Instruktion `WaitSensor` sind unten aufgeführt.

Siehe auch *Weitere Beispiele auf Seite 1086*.

Beispiel 1:

```
WaitSensor Ssync1;
```

Das Programm baut eine Verbindung zum ersten Objekt in der Objektqueue auf, die sich innerhalb des Startfensters auf dem Sensor befindet. Wenn sich im Startfenster keine Objekte befinden, wird die Abarbeitung angehalten und das Programm wartet auf ein Objekt.

Argumente

```
WaitSensor MechUnit [\RelDist ][\PredTime][\MaxTime][\TimeFlag]
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Die sich bewegende mechanische Einheit, auf die sich die Roboterposition in der Instruktion bezieht.

`[\RelDist]`

Relative Entfernung

Datentyp: `num`

Wartet auf das Erscheinen eines Objekts im Startfenster, das den im Argument angegebenen Abstand überschreitet. Wenn das Werkobjekt bereits verbunden ist, wird die Abarbeitung angehalten, bis das Objekt den angegebenen Abstand passiert hat. Wenn sich das Objekt bereits hinter den relativen Abstand bewegt hat, wird die Abarbeitung fortgesetzt.

`[\PredTime]`

Vorlaufzeit

Datentyp: `num`

Wartet auf das Erscheinen eines Objekts im Startfenster, das den im Argument angegebenen Abstand überschreitet. Wenn das Werkobjekt bereits verbunden ist, wird die Abarbeitung angehalten, bis das Objekt den angegebenen Abstand passiert hat. Wenn das Objekt die Vorlaufzeit bereits überschritten hat, wird die Abarbeitung fortgesetzt.

`[\MaxTime]`

Maximale Wartezeit

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.332 WaitSensor - Auf die Sensorverbindung warten
Machine Synchronization
Fortsetzung

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor eine Sensorverbindung aufgebaut oder \RelDist erreicht ist, wird die Fehlerbearbeitung mit dem Fehlercode ERR_WAIT_MAXTIME aufgerufen, vorausgesetzt, es wurde eine definiert. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

[\TimeFlag]

Timeout-Flag

Datentyp: bool

Der Ausgabeparameter, der den Wert TRUE enthält, wenn die maximal zulässige Dauer für die Wartezeit abläuft, bevor die Sensorverbindung oder \RelDist erzielt wird. Wird dieser Parameter in der Instruktion verwendet, wird kein Fehler ausgegeben, wenn die max. Dauer abläuft.

Dieses Argument wird ignoriert, wenn das Argument MaxTime nicht in der Instruktion enthalten ist.

Programmabarbeitung

Wenn sich im Startfenster kein Objekt befindet, wird die Programmabarbeitung angehalten. Ist ein Objekt vorhanden, wird das Objekt mit dem Sensor verbunden und die Abarbeitung wird fortgesetzt.

Wird eine zweite WaitSensor-Instruktion bei vorhandener Verbindung ausgegeben, führt das zu einer Fehlermeldung, wenn das optionale Argument \RelDist nicht verwendet wird.

Fehlerbehandlung

Wenn die folgenden Fehler während der Abarbeitung der Instruktion WaitSensor auftreten, wird die Systemvariable ERRNO festgelegt. Diese Fehler können daraufhin in der Fehlerbearbeitung behoben werden.

Name	Fehlerursache
ERR_CNV_NOT_ACT	Der Sensor ist nicht aktiviert.
ERR_CNV_CONNECT	Die Instruktion WaitSensor ist bereits verbunden.
ERR_CNV_DROPPED	Das von der Instruktion WaitSensor erwartete Objekt wurde von einem anderen Task gelöscht.
ERR_WAIT_MAXTIME	Das Objekt hat die Wartezeit überschritten und es ist kein TimeFlag vorhanden.
ERR_CNV_OBJ_LOST	Das Objekt hat StartwindowWidth ohne verbunden zu werden passiert.

Weitere Beispiele

Weitere Beispiele für die Instruktion sind unten aufgeführt.

Beispiel 1:

```
WaitSensor Ssync1\RelDist:=500.0;
```

Bei fehlender Verbindung muss gewartet werden, bis das Objekt in das Startfenster gelangt und dann den 500-mm-Punkt am Sensor passiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Falls bereits eine Verbindung zum Objekt besteht, wird darauf gewartet, dass das Objekt den 500-mm-Punkt passiert.

Beispiel 2:

```
WaitSensor Ssync1\RelDist:=0.0;
```

Wenn keine Verbindung besteht, wird darauf gewartet, dass ein Objekt in das Startfenster bewegt wird.

Wenn bereits eine Verbindung besteht, wird die Abarbeitung fortgesetzt, da das Objekt bereits den 0,0-mm-Punkt passiert hat.

Beispiel 3

```
WaitSensor Ssync1;
WaitSensor Ssync1\RelDist:=0.0;
```

Der erste `WaitSensor` baut eine Verbindung zum Objekt im Startfenster auf. Der zweite `WaitSensor` kehrt sofort zurück, wenn das Objekt immer noch verbunden ist, aber er wartet darauf, dass das nächste Objekt hinter die maximale Distanz bewegt oder entfernt wird.

Beispiel 4:

```
WaitSensor Ssync1\RelDist:=0.5\PredTime:=0.1;
```

Der `WaitSensor` kehrt sofort zurück, wenn das Objekt den Punkt 0,5 Meter überschritten hat. Ist das nicht der Fall, wartet er darauf, bis ein Objekt *=RelDist - C1speed * PredTime* erreicht. Hier besteht der Zweck darin, die Verzögerungen vorherzusagen, bevor eine neue Bewegungsinstruktion gestartet wird.

Beispiel 5:

```
WaitSensor Ssync1\RelDist:=0.5\MaxTime:=0.1\TimeFlag:=flag1;
```

Der `WaitSensor` kehrt sofort zurück, wenn das Objekt den Punkt 0,5 Meter überschritten hat. Ist das nicht der Fall, wartet er 0,1 s auf ein Objekt. Wenn kein Objekt den Punkt 0,5 Meter in diesen 0,1 s passiert, kehrt die Instruktion mit `flag1 = TRUE` zurück.

Einschränkungen

Es dauert 50 ms, bis die Verbindung zum ersten Objekt im Startfenster aufgebaut wird. Sobald eine Verbindung besteht, benötigt der zweite `WaitSensor` mit dem optionalen Argument `\RelDist` nur die normale Abarbeitungszeit für die Instruktion RAPID.

Syntax

```
WaitSensor
  [ MechUnit ':=' ] < variable (VAR) of mecunit >
  [ '\' RelDist ':=' < expression (IN) of num > ]
  [ '\' PredTime ':=' < expression (IN) of num > ]
  [ '\' MaxTime ':=' < expression (IN) of num > ]
  [ '\' TimeFlag ':=' < variable (VAR) of bool > ] ';'
```

Weitere Informationen

Information über	Siehe
Ablegen des Objekts am Sensor	*DropSensor - Ablegen des Objekts am Sensor auf Seite 191*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.332 WaitSensor - Auf die Sensorverbindung warten

Machine Synchronization

Fortsetzung

Information über	Siehe
Synchr. mit Sensor	*SyncToSensor - Synchr. mit Sensor auf Seite 875*
Synchr. mit Sensor	*SyncToSensor - Synchr. mit Sensor auf Seite 875*
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.333 WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten

Verwendung

`WaitSyncTask` wird verwendet, um mehrere Programmtasks an einem bestimmten Punkt im Programm zu synchronisieren. Jede Programmtask wartet, bis alle Programmtasks den angegebenen Synchronisierungspunkt erreicht haben.

Hinweis

Die `WaitSyncTask`-Instruktion synchronisiert nur die Programmabarbeitung. Um sowohl die Programmabarbeitung als auch die Roboterbewegungen zu synchronisieren, muss die `Move`-Instruktion vor der `WaitSyncTask` in allen beteiligten Tasks ein Stopppunkt sein. Die Programmabarbeitung und die Roboterbewegungen können auch unter Verwendung von `WaitsyncTask \Inpos ...` in allen beteiligten Programmen synchronisiert werden.

WARNUNG

Für eine sichere Synchronisierungsfunktionalität muss der Treffpunkt (Parameter `SyncID`) in jeder Programmtask über einen eindeutigen Namen verfügen. Der Name muss auch für die Programmtasks übereinstimmen, die sich im Treffpunkt treffen sollen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WaitSyncTask`:

Siehe auch *Weitere Beispiele auf Seite 1092*.

Beispiel 1:

Programmbeispiel in Task `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

...
WaitSyncTask sync1, task_list;
...
```

Beispiel 2:

Programmbeispiel in Task `T_ROB2`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

...
WaitSyncTask sync1, task_list;
...
```

Die Programmtask, die zuerst `WaitSyncTask` mit der Identität `sync1` erreicht, wartet, bis die andere Programmtask `WaitSyncTask` mit derselben Identität `sync1`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.333 WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten
Multitasking
Fortsetzung

erreicht. Dann fahren die beiden Programmtasks T_ROB1 und T_ROB2 mit der Abarbeitung fort.

Argumente

```
WaitSyncTask [\InPos] SyncID TaskList [\TimeOut]
```

[\InPos]

In Position

Datentyp: switch

Wenn dieses Argument verwendet wird, müssen Roboter und externe Achsen zum Stillstand gekommen sein, bevor diese Programmtask wartet, bis andere Programmtasks den in der Instruktion WaitSyncTask angegebenen Treffpunkt erreichen.

SyncID

Synchronization identity

Datentyp: syncident

Variable, die den Namen des Synchronisierungspunkts (Treffpunkts) angibt. Der Datentyp syncident ist ein Datentyp ohne einen Wert, der nur als Bezeichner für den Synchronisierungspunkt fungiert.

Die Variable muss definiert werden und in allen koordinierten Programmtasks den gleichen Namen besitzen. Es wird empfohlen, die Variable immer global in jeder Programmtask zu definieren (VAR syncident ...).

TaskList

Datentyp: tasks

Persistente Variable, die in einer Taskliste (einem Datenfeld) den Namen (string) der Programmtasks angibt, die am Synchronisierungspunkt mit dem vom Argument SyncID angegebenen Namen warten sollen.

Die persistente Variable muss definiert werden und in allen koordinierten Programmtasks über den gleichen Namen sowie den gleichen Inhalt verfügen. Es wird empfohlen, die Variable immer global im System zu definieren (PERS tasks ...).

[\TimeOut]

Datentyp: num

Maximale Dauer, für die am Synchronisierungspunkt auf die anderen Programmtasks gewartet wird. Timeout in Sekunden (Auflösung 0,001 s). Wenn dieses Argument nicht angegeben ist, wartet die Programmtask für unbegrenzte Zeit.

Wenn diese Zeit abläuft, bevor die Programmtasks den Synchronisierungspunkt erreicht haben, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode ERR_WAITSYNCTASK aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.333 WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten

Multitasking
Fortsetzung

Programmabarbeitung

Die aktuelle Programmtask wartet bei `WaitSyncTask`, bis die anderen Programmtasks in der `TaskList` denselben `SyncID`-Punkt erreicht haben. Dann fahren die jeweiligen Programmtasks mit der Abarbeitung der nächsten Instruktion fort.

`WaitSyncTask` kann zwischen Bewegungsinstruktionen mit dazwischen liegenden Verschleifzonen programmiert werden. Je nach Zeitspanne zwischen den Programmtasks bei Abarbeitungszeit kann das System:

- bei optimaler Abstimmung alle Verschleifzonen halten.
- bei schlechtester Abstimmung nur die Verschleifzone der Programmtask halten, die `WaitSyncTask` als letzte erreicht. In den anderen Programmtasks führt dies zu Stopppunkten.

Programmtasks können zum Testen über das FlexPendant-Taskauswahlfeld ausgeschlossen werden.

Die folgenden Prinzipien können verwendet werden:

- Prinzip 1) Schließen Sie die Programmtask für den gesamten Zyklus im Taskauswahlfeld aus, bevor Sie von „main“ starten (nach dem Setzen des `PP` auf „main“) - Dieser Ausschluss ist während des gesamten Programmzyklus gültig.
- Prinzip 2) Schließen Sie die Programmtask vorübergehend im Taskauswahlfeld zwischen `WaitSyncTask`-Instruktionen im Programmzyklus aus. - Das System führt nur die anderen verbundenen Tasks aus, doch der Benutzer wird mit einer Fehlermeldung gezwungen, die ausgeschlossenen Programmtasks zu verbinden, bevor die koordinierte `WaitSyncTask` übergeben wird.
- Prinzip 3) Wenn nach Prinzip 2 vorgegangen wird, können einige Programmtasks für den gesamten Zyklus im Taskauswahlfeld ausgeschlossen werden, um weiter gemäß Prinzip 1 vorzugehen, indem die Serviceroutine `SkipTaskExec` ausgeführt wird.

Beachten Sie, dass das Taskauswahlfeld gesperrt ist, wenn das System mit synchronisierten Bewegungen ausgeführt wird.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_WAITSYNCTASK`	Es wird ein Timeout erreicht, weil `WaitSyncTask` nicht bereit ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.333 WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten

Multitasking
Fortsetzung

Weitere Beispiele

Weitere Beispiele für die Instruktion `WaitSyncTask` sind unten aufgeführt.

Beispiel 1:

Programmbeispiel in Task `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

...
WaitSyncTask \InPos, sync1, task_list \TimeOut := 60;
...
ERROR
  IF ERRNO = ERR_WAITSYNCTASK THEN
    RETRY;
  ENDIF
```

Die Programmtask `T_ROB1` wartet in der Instruktion `WaitSyncTask`, bis sich die mechanischen Einheiten in Position befinden, und wartet anschließend, bis die Programmtask `T_ROB2` den Synchronisierungspunkt mit derselben Identität erreicht. Nach einer Wartezeit von `60` s wird die Fehlerbehandlungsroutine mit `ERRNO` gleich `ERR_WAITSYNCTASK` aufgerufen. Dann wird die Instruktion `WaitSyncTask` nochmals aufgerufen und die Wartezeit um 60 s verlängert.

Einschränkung

Wenn dieser Instruktion eine Bewegungsinstruktion vorangeht, muss diese mit einem Stopppunkt (zonedata `fine`) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`WaitSyncTask \InPos` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop,Restart oder Step.

Syntax

```
WaitSyncTask
  ['\' InPos ',']
  [ SyncID ':=' ] < variable (VAR) of syncident> ','
  [ TaskList ':=' ] < persistent array {*} (PERS) of tasks>
  [ '\' TimeOut ':=' < expression (IN) of num > ] ';'
```

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Identität für Synchronisierungspunkte	*syncident - Identität für Synchronisierungspunkte auf Seite 1814*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.334 WaitTestAndSet – Warten, bis Variable FALSE wird, dann einstellen

Verwendung

Die Instruktion `WaitTestAndSet` wartet darauf, dass eine angegebene persistente `bool`-Variable den Wert `FALSE` annimmt. Sobald die Variable `FALSE` ist, setzt die Instruktion den Wert auf `TRUE` und fährt mit der Abarbeitung fort. Die persistente Variale kann als binäres Semaphor zur Synchronisierung und zum gegenseitigen Ausschluss verwendet werden.

Dieser Instruktion liegt dieselbe Funktionalität zugrunde wie der Funktion `TestAndSet`. Die Instruktion `WaitTestAndSet` wartet jedoch, bis der Wert der `bool`-Variablen `FALSE` ist, während die Instruktion `TestAndSet` sofort die Abarbeitung beendet.

Es wird nicht empfohlen, die Instruktion `WaitTestAndSet` in Interrupt-Routinen, UNDO-Behandlungsroutinen oder Ereignisroutinen zu verwenden.

Beispiele für Ressourcen, die möglicherweise vor gleichzeitigem Zugriff geschützt werden müssen:

- RAPID-Routinen mit Funktionsproblemen bei simultaner Abarbeitung
- Vewendung des FlexPendant - Bedienerprotokoll

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitTestAndSet`:

Siehe auch *Weitere Beispiele auf Seite 1094*.

Beispiel 1:

`MAIN`-Programmtask:

```
PERS bool tproutine_inuse := FALSE;
...
WaitTestAndSet tproutine_inuse;
TPWrite "First line from MAIN";
TPWrite "Second line from MAIN";
TPWrite "Third line from MAIN";
tproutine_inuse := FALSE;
```

`BACK1`-Programmtask:

```
PERS bool tproutine_inuse := FALSE;
...
WaitTestAndSet tproutine_inuse;
TPWrite "First line from BACK1";
TPWrite "Second line from BACK1";
TPWrite "Third line from BACK1";
tproutine_inuse := FALSE;
```

Damit die Zeilen im Bedienerprotokoll (eine Zeile aus `MAIN` und eine Zeile aus `BACK1`) nicht verwechselt werden, verhindert die Funktion `WaitTestAndSet`, dass die drei Zeilen aus jeder Task getrennt werden.

Wenn die Programmtask `MAIN` das Semaphor `WaitTestAndSet(tproutine_inuse)` zuerst annimmt, muss die Programmtask `BACK1` warten, bis die Programmtask `MAIN` das Semaphor verlassen hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.334 WaitTestAndSet – Warten, bis Variable FALSE wird, dann einstellen

RobotWare - OS

Fortsetzung

Argumente

```
WaitTestAndSet Object
```

`Object`

Datentyp: `bool`

Anwenderdefiniertes Datenobjekt, das als Semaphor verwendet werden soll. Das Datenobjekt muss eine persistente Variable `PERS` sein. Wenn `WaitTestAndSet` für verschiedene Programmtasks verwendet wird, muss es sich um ein globales `PERS`-Objekt handeln.

Programmabarbeitung

Diese Instruktion überprüft in einem unteilbaren Schritt die benutzerdefinierte persistente Variable und legt sie wie in folgendem Codebeispiel fest:

- Wenn der Wert `FALSE` ist, wird er auf `TRUE` gesetzt.
- Wenn der Wert `TRUE`, ist, wartet die Instruktion, bis er `FALSE` wird, und setzt ihn dann auf `TRUE`

```
IF Object = FALSE THEN
  Object := TRUE;
ELSE
  ! Wait until it become FALSE
  WaitUntil Object = FALSE;
  Object := TRUE;
ENDIF
```

Im Anschluss daran ist die Instruktion abgeschlossen. Um Probleme zu vermeiden, die auftreten können, weil persistente Variablen ihren Wert behalten, wenn der Programmzeiger PZ auf „Main“ gesetzt wird, sollte das Semaphor-Objekt in der `START`-Ereignisroutine immer auf `FALSE` gesetzt werden.

Weitere Beispiele

Weitere Beispiele für die Instruktion `WaitTestAndSet` sind unten aufgeführt.

Beispiel 1:

```
PERS bool semPers:= FALSE;
...
PROC doit(...)
  WaitTestAndSet semPers;
  ...
  semPers := FALSE;
ENDPROC
```

Hinweis

Wenn die Programmabarbeitung in der Routine `doit` gestoppt wird und der Programmzeiger auf `main` verschoben wird, wird die Variable `semPers` nicht zurückgesetzt. Um dies zu vermeiden, sollte die Variable `semPers` in der `START`-Ereignisroutine auf `FALSE` gesetzt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.334 WaitTestAndSet – Warten, bis Variable FALSE wird, dann einstellen

Syntax

```
WaitTestAndSet
  [ Object ':=' ] < persistent (PERS) of bool> ';'
```

Weitere Informationen

Information über	Siehe
Testen und ggf. Setzen einer nicht gesetzten Variablen (Typabfrage mit WaitTime)	*TestAndSet - Prüfen und ggf. Setzen einer nicht gesetzten Variablen auf Seite 1549*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.335 WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist

Verwendung

`WaitTime` wird verwendet, um eine vorgegebene Zeit zu warten. Mithilfe dieser Instruktion kann gewartet werden, bis der Roboter und externe Achsen zum Stillstand gekommen sind.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitTime`:

Siehe auch *Weitere Beispiele auf Seite 1096* unten.

Beispiel 1:

```
WaitTime 0.5;
```

Die Programmabarbeitung wartet 0,5 Sekunden.

Argumente

```
WaitTime [\InPos] Time
```

`[\InPos]`

In Position

Datentyp: `switch`

Wenn dieses Argument verwendet wird, müssen Roboter und externe Achsen zum Stillstand gekommen sein, bevor die Wartezeit beginnt. Dieses Argument kann nur verwendet werden, wenn die Task mechanische Einheiten steuert.

`Time`

Datentyp: `num`

Die Zeitdauer in Sekunden, für die die Programmabarbeitung warten soll. Der Mindestwert ist 0 s. Es gibt keinen Höchstwert. Auflösung ist 0,001 s.

Programmabarbeitung

Die Programmabarbeitung hält für die vorgegebene Zeit an. Interrupt-Behandlung und ähnliche Funktionen sind immer noch aktiv.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `WaitTime` sind unten aufgeführt.

Beispiel 1:

```
WaitTime \InPos,0;
```

Die Programmabarbeitung wartet, bis der Roboter und die externen Achsen zum Stillstand gekommen sind.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Das Argument `\Inpos` kann nicht in Kombination mit *SoftServo* verwendet werden.

Wenn dieser Instruktion eine`Move`-Instruktion vorangeht, muss diese `Move`-Instruktion mit einem Stopppunkt (zonedata fine) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

`WaitTime \Inpos` kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

Syntax

```
WaitTime
  ['\' InPos ',']
  [ Time ':='] <expression (IN) of num> ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis eine Bedingung erfüllt ist	*WaitUntil – Warten, bis eine Bedingung erfüllt ist auf Seite 1098*
Warten, bis ein E/A gesetzt/zurückgesetzt wird	*WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist auf Seite 1052*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.336 WaitUntil – Warten, bis eine Bedingung erfüllt ist

Verwendung

`WaitUntil` wird verwendet, um zu warten, bis eine logische Bedingung erfüllt ist. Beispielsweise kann gewartet werden, bis einer oder mehrere Eingänge gesetzt wurden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitUntil`:

Siehe auch *Weitere Beispiele auf Seite 1101*.

Beispiel 1:

```
WaitUntil di4 = 1;
```

Die Programmabarbeitung fährt erst fort, nachdem der Eingang `di4` gesetzt wurde.

Argumente

```
WaitUntil [\InPos] Cond [\MaxTime] [\TimeFlag] [\PollRate]
     [\Visualize] [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon] [\Image] [\VisualizeTime] [\UIActiveSignal]
```

`[\InPos]`

In Position

Datentyp: `switch`

Wenn dieses Argument verwendet wird, müssen der Roboter und die externen Achsen den Stopppunkt (`ToPoint` der aktuellen Bewegungsinstruktion) erreicht haben, bevor die Abarbeitung fortgesetzt werden kann. Dieses Argument kann nur verwendet werden, wenn die Task mechanische Einheiten steuert.

`Cond`

Datentyp:`bool`

Der logische Ausdruck, auf den gewartet werden soll.

`[\MaxTime]`

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Bedingung erfüllt ist, wird die Fehlerbehandlungsroutine (falls vorhanden) mit dem Fehlercode `ERR_WAIT_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

`[\TimeFlag]`

Timeout Flag

Datentyp: `bool`

Der Ausgabeparameter, der den Wert TRUE enthält, wenn die maximal zulässige Wartezeit abläuft, bevor die Bedingung erfüllt wird. Wenn dieser Parameter in der Instruktion angegeben wird, wird das Ablaufen der maximalen Zeit nicht als Fehler betrachtet. Dieses Argument wird ignoriert, wenn das Argument `MaxTime` nicht in der Instruktion angegeben wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\PollRate]`

Polling Rate

Datentyp: `num`

Die Abfragerate in Sekunden, um zu überprüfen, ob die Bedingung im Argument `Cond` den Wert `TRUE` aufweist. Das bedeutet, dass `WaitUntil` die Bedingung sofort überprüft, und bei einem anderen Wert als `TRUE` die Überprüfung mit der angegebenen Abfragerate wiederholt, bis die Bedingung `TRUE` ist. Der Mindestwert für die Abfragerate ist 0,004 s. Wenn dieses Argument nicht verwendet wird, ist die Standardabfragerate auf 0,1 s gesetzt.

`[\Visualize]`

Datentyp: `switch`

Wenn ausgewählt ist, wird die Visualisierung aktiviert. Die Visualisierung besteht aus einem Dialogfenster mit der logischen Bedingung, die nicht erfüllt ist, Symbol, Kopfzeile, Meldezeilen. Es wird ein Bild gemäß den programmierten Argumenten angezeigt.

`[\Header]`

Datentyp: `string`

Der Kopftext muss oben im Nachrichtenfenster eingegeben werden. Maximal40 Zeichen. Wenn kein`\Header` Argument eingegeben wird, wird eine Standardnachricht angezeigt.

`[\Message]`

Datentyp: `string`

Es muss eine Textzeile im Bildschirm eingegeben werden. Maximal 50 Zeichen.

`[\MsgArray]`

(Message Array)

Datentyp: `string`

Auf dem Bildschirm müssen mehrere Textzeilen eines Bereichs eingegeben werden. Es kann nur einer der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der höchstzulässige Raum für das Layout ist 5 Zeilen mit jeweils 50 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile auf dem Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1712*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Standardmäßig wird kein Icon angezeigt.

`[\Image]`

Datentyp: `string`

Der Name des zu verwendenden Bildes. Um eigene Bilder zu verwenden, müssen die Bilder im Verzeichnis *HOME:* des aktiven Systems oder direkt im aktiven System abgelegt sein.

Es wird empfohlen, die Dateien im Verzeichnis *HOME:* abzulegen, so dass sie bei einem Backup und einer Wiederherstellung gespeichert werden.

Ein Neustart muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

`[\VisualizeTime]`

Datentyp: `num`

Die Wartezeit, bevor das Dialogfeld auf dem FlexPendant erscheinen soll. Wenn die Argumente `\VisualizeTime` und `\MaxTime` verwendet werden, muss die im Argument `\MaxTime` verwendete Zeit größer sein als die Zeit, die im Argument `\VisualizeTime` verwendet wird.

Die Standardzeit für die Visualisierung, ohne Argument `\VisualizeTime` ist 5 s. Minimalwert 1 s. Maximalwert ohne Begrenzung. Auslösung 0.001 s.

`[\UIActiveSignal]`

Datentyp: `signaldo`

Das digitale Ausgangssignal, das im optionalen Argument `UIActiveSignal`verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn das Meldungsfenster entfernt wird (wenn die Bedingung erfüllt ist), wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Instruktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Wenn die programmierte Bedingung bei der Abarbeitung einer `WaitUntil`-Instruktion nicht erfüllt wird, wird die Bedingung alle 100 ms erneut geprüft (oder gemäß dem Wert im Argument `Pollrate`).

Die Wartezeit des Roboters wird überwacht. Standardmäßig kann der Roboter für eine unbegrenzte Zeitspanne warten, doch die maximale Wartezeit kann mit dem

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

optionalen Argument \MaxTime angegeben werden. Wenn diese maximale Zeit überschritten wird, wird ein Fehler ausgelöst.

Im manuellen Modus wird nach einer Wartezeit von mehr als3 s eine Warnmeldung angezeigt, in der Sie gefragt werden, ob die Anweisung simuliert werden soll. Sie können die Warnung so konfigurieren, dass sie nicht angezeigt wird, indem Sie die Systemparameter *SimulateMenu* auf NO setzen. Siehe *Technisches Referenzhandbuch - Systemparameter*, Thema *Controller*, Typ *General RAPID*.

Wenn der Schalter \Visualize verwendet wird, erscheint auf dem FlexPendant ein Dialogfeld entsprechend den programmierten Argumenten. Wird kein \Header-Argument verwendet, wird ein Standardkopfzeilentext verwendet. Wenn die Abarbeitung der WaitUntil-Instruktion beendet ist, wird das Dialogfeld vom FlexPendant entfernt.

Neues Meldungsfenster auf TRAP-Ebene bringt den Fokus vom Dialogfenster auf die Basisebene.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es gibt ein Signal, das in der Bedingung verwendet wird, und es besteht kein Kontakt zu dem E/A-Gerät.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
ERR_WAIT_MAXTIME	Es liegt ein Timeout vor (Parameter \MaxTime), bevor die Bedingung zum richtigen Wert gewechselt ist.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion WaitUntil sind unten aufgeführt.

Beispiel 1:

```
VAR bool timeout;
WaitUntil start_input = 1 AND grip_status = 1\MaxTime := 60
     \TimeFlag := timeout;
IF timeout THEN
  TPWrite "No start order received within expected time";
ELSE
  start_next_cycle;
ENDIF
```

Wenn die beiden Eingabebedingungen nicht innerhalb von 60 Sekunden erfüllt werden, wird am FlexPendant eine Fehlermeldung ausgegeben.

Beispiel 2:

```
WaitUntil \Inpos, di4 = 1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.336 WaitUntil – Warten, bis eine Bedingung erfüllt ist

RobotWare - OS

Fortsetzung

Die Programmabarbeitung wartet, bis der Roboter zum Stillstand gekommen ist und der Eingang `di4` gesetzt wurde.

Beispiel 3

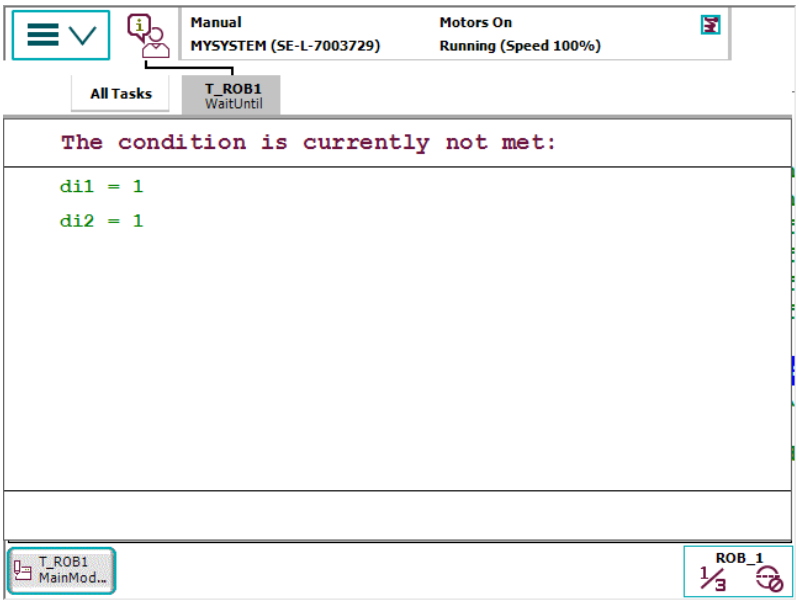
```
WaitUntil di4 = 1 \MaxTime:=5.5;
..
ERROR
  ERROR
    IF ERRNO = ERR_WAIT_MAXTIME THEN
      RAISE;
    ELSE
      Stop;
    ENDIF
```

Die Programmabarbeitung wird ausgesetzt, bis der Eingang `di4` gesetzt wurde. Wenn das E/A-Gerät deaktiviert wurde oder wenn die Wartezeit abgelaufen ist, wird die Abarbeitung in der Fehlerbehandlung fortgesetzt.

Beispiel 4:

```
WaitUntil di1 = 1 AND di2 = 1 \MaxTime := 60 \Visualize;
..
  ERROR
    IF ERRNO = ERR_WAIT_MAXTIME THEN
      RAISE;
    ELSE
      Stop;
    ENDIF
```

Wenn die beiden Eingangsbedingungen nicht innerhalb von`5` Sekunden erfüllt werden, erscheint eine Meldung auf dem Display des FlexPendant. Wenn die Bedingungen nicht innerhalb von `60` erfüllt werden, wird die Abarbeitung in der Fehlerbehandlung fortgesetzt.



xx1600000146

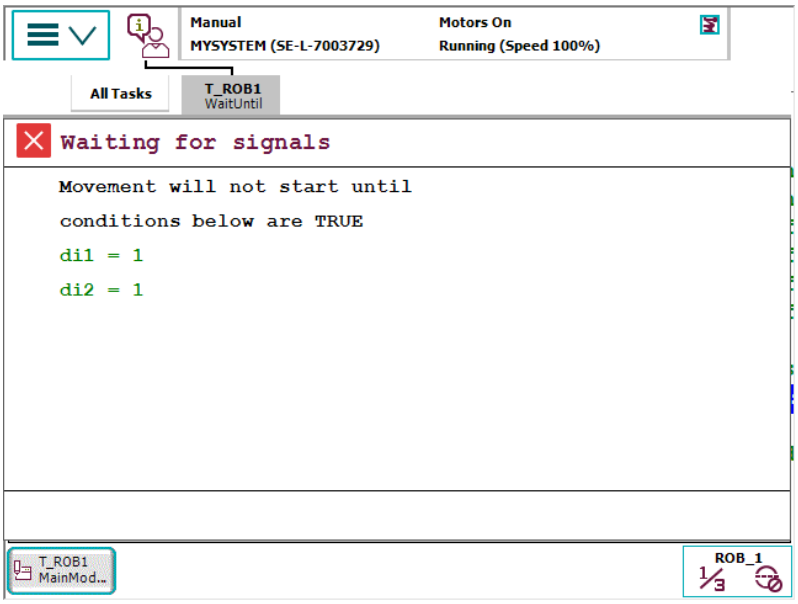
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 5:

```
WaitUntil di1 = 1 AND di2 = 1 \Visualize \Header:="Waiting for
     signals" \MsgArray:=["Movement will not start until",
     "conditions below are TRUE"] \Icon:=iconError;
MoveL p40, v500, z20, L10tip;
..
```

Wenn die zwei Eingabebedingungen nicht erfüllt werden, werden die Kopfzeile und die Nachricht in den optionalen Bemerkungen `\Header` wiedergegeben und `\MsgArray` wird auf der Anzeige des FlexPendant gemeinsam mit den Bedingungen, die nicht erfüllt wurden, angezeigt.



xx1600000147

Beispiel 6:

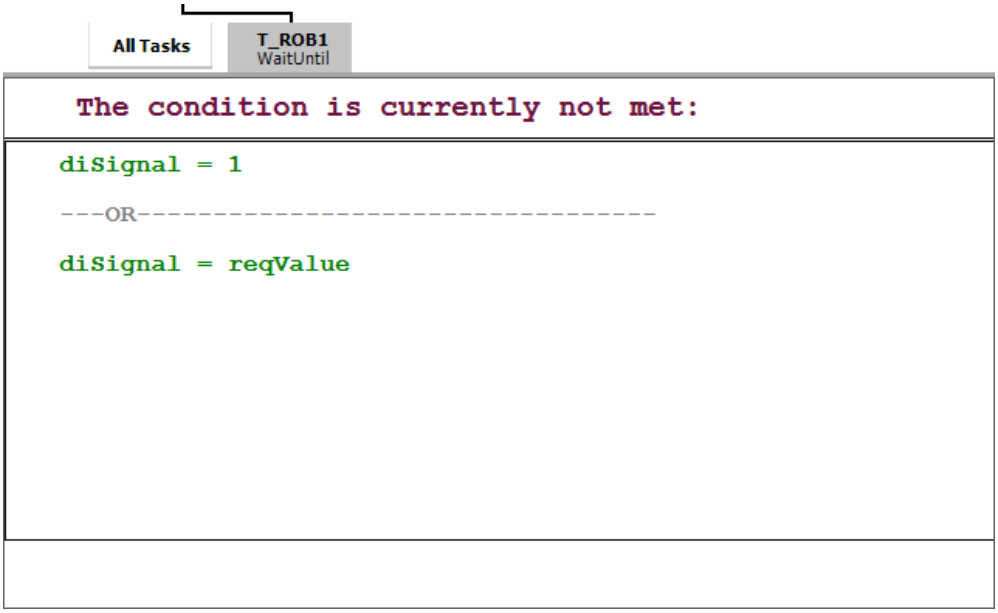
```
VAR num reqValue:=1;
Waituntil diSignal = 1 OR diSignal = reqValue\Visualize;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.336 WaitUntil – Warten, bis eine Bedingung erfüllt ist
RobotWare - OS
Fortsetzung

Wenn eine Variable in der Bedingung mit \Visualize verwendet wird, dann wird die Variable angezeigt, und nicht deren Wert, wenn die Eingangsbedingungen nicht erfüllt sind.



xx2100000553

Einschränkung

Das Argument \Inpos kann nicht in Kombination mit *SoftServo* verwendet werden.

Wenn dieser Instruktion eineMove-Instruktion vorangeht, muss diese Move-Instruktion mit einem Stopppunkt (zonedata fine) und nicht mit einem Fly-By-Punkt programmiert werden. Anderenfalls ist ein Neustart nach einem Stromausfall nicht möglich.

WaitUntil \Inpos kann nicht in einer RAPID-Routine ausgeführt werden, die mit einem der folgenden speziellen Systemereignisse verknüpft ist: PowerOn, Stop, QStop, Restart oder Step.

WaitUntil \Inpos kann nicht in Kombination mit StopMove verwendet werden, um festzustellen, ob die Bewegung angehalten wurde. Die WaitUntil-Instruktion kann in diesem Fall für unbeschränkte Zeit hängen bleiben. Es wird nicht erkannt, das die Bewegung angehalten wurde. Es wird erkannt, dass der Roboter und die externen Achsen den letzten programmierten ToPoint (MoveX, SearchX, TriggX) erreicht haben.

Syntax

```
WaitUntil
  ['\' InPos ',']
  [Cond ':='] <expression (IN) of bool>
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' TimeFlag ':=' <variable (VAR) of bool>]
  ['\' PollRate ':=' <expression (IN) of num>]
  ['\' Visualize]
  ['\' Header ':=' <expression (IN) of string>]]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
['\' Message ':=' <expression (IN) of string>]
| ['\' MsgArray ':=' <array {*} (IN) of string>]
['\' Wrap]
['\' Icon ':=' <expression (IN) of icondata>]
['\' Image ':=' <expression (IN) of string>]
['\' VisualizeTime ':=' <expression (IN) of num>]
['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ';'
```

Weitere Informationen

Information über	Siehe
Warten, bis ein Eingang gesetzt/zurückgesetzt wird	*WaitDI - Warten, bis ein digitales Eingangssignal gesetzt ist auf Seite 1052*
Warten über einen angegebenen Zeitraum	*WaitTime – Warten, bis ein vorgegebener Zeitraum verstrichen ist auf Seite 1096*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.337 WaitWObj - Warten auf Werkobjekt am Förderer

Verwendung

`WaitWObj` (*Wait Work Object*) stellt die Verbindung zu einem Werkobjekt im Startfenster an einer mechanischen Fördereinheit her.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WaitWObj`:

Siehe auch *Weitere Beispiele auf Seite 1107*.

Beispiel 1:

```
WaitWObj wobj_on_cnv1;
```

Das Programm baut eine Verbindung zum ersten Objekt in der Objektqueue auf, das sich im Startfenster am Conveyor befindet. Ist kein Objekt im Startfenster vorhanden, wartet die Abarbeitung auf ein Objekt.

Argumente

```
WaitWObj WObj [ \RelDist ][\MaxTime][\TimeFlag]
```

`WObj`

Work Object

Datentyp: `wobjdata`

Das sich bewegende Werkobjekt (Koordinatensystem), auf das sich die Roboterposition in der Instruktion bezieht. Die mechanische Einheit des Conveyors muss durch die Komponente `ufmec` im Werkobjekt angegeben werden.

`[ \RelDist ]`

Relative Distance

Datentyp: `num`

Warten auf das Erscheinen eines Objekts im Startfenster, das den im Argument angegebenen Abstand überschreitet. Wenn das Werkstück bereits verbunden ist, wartet die Abarbeitung, bis es die angegebene Entfernung überschreitet. Wenn das Objekt `\RelDist` bereits überschritten hat, wird die Abarbeitung fortgesetzt.

`[\MaxTime]`

Maximum Time

Datentyp: `num`

Die maximal zulässige Wartezeit in Sekunden. Wenn diese Zeit abläuft, bevor die Objektverbindung oder `\Reldist` erreicht wird, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_WAIT_MAXTIME` aufgerufen (sofern vorhanden). Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

`[\TimeFlag]`

Timeout Flag

Datentyp: `bool`

Der Ausgabeparameter, der den Wert `TRUE` enthält, wenn die maximal zulässige Wartezeit abläuft, bevor die Sensorverbindung aufgebaut oder `\Reldist` erreicht

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wurde. Wenn dieser Parameter in der Instruktion angegeben wird, wird das Ablaufen der maximalen Zeit nicht als Fehler betrachtet. Dieses Argument wird ignoriert, wenn das Argument `MaxTime` nicht in der Instruktion angegeben wird.

Programmabarbeitung

Falls kein Objekt im Startfenster vorhanden ist, wartet die Programmabarbeitung. Befindet sich dort jedoch ein Objekt, wird das Werkobjekt mit dem Conveyor verbunden und die Abarbeitung fortgesetzt.

Wird eine zweite `WaitWObj`-Instruktion bei vorhandener Verbindung ausgegeben, führt das zu einer Fehlermeldung, wenn das optionale Argument `\RelDist` nicht verwendet wird.

Fehlerbehandlung

Bei Auftreten folgender Fehler während der Abarbeitung der `WaitWobj`-Instruktion wird die Systemvariable `ERRNO` gesetzt. Diese Fehler können dann in der Fehlerbehandlungsroutine behoben werden.

Name	Fehlerursache
`ERR_CNV_NOT_ACT`	Der Conveyor ist nicht aktiviert.
`ERR_CNV_CONNECT`	Die Instruktion `WaitWObj` ist bereits verbunden.
`ERR_CNV_DROPPED`	Das von der Instruktion `WaitWObj` erwartete Objekt wurde von einem anderen Task gelöscht.
`ERR_WAIT_MAXTIME`	Das Objekt hat die Wartezeit überschritten und es ist kein `Timeflag` vorhanden.
`ERR_CNV_OBJ_LOST`	Das Objekt hat `StartwindowWidth` ohne verbunden zu werden passiert.

Weitere Beispiele

Weitere Beispiele für die Instruktion `WaitWObj` sind unten aufgeführt.

Beispiel 1:

```
WaitWObj wobj_on_cnv1\RelDist:=500.0;
```

Bei fehlender Verbindung muss gewartet werden, bis das Objekt in das Startfenster gelangt und dann den 500-mm-Punkt am Conveyor passiert.

Bei bestehender Verbindung zum Objekt muss gewartet werden, bis das Objekt den 500-mm-Punkt passiert hat.

Bei fehlender Verbindung wird auf ein Objekt im Startfenster gewartet.

Beispiel 2:

```
WaitWObj wobj_on_cnv1\RelDist:=0.0;
```

Bei bestehender Verbindung wird die Abarbeitung fortgesetzt, da das Objekt den Punkt 0,0 mm bereits passiert hat.

Beispiel 3

```
WaitWObj wobj_on_cnv1;
WaitWObj wobj_on_cnv1\RelDist:=0.0;
```

Die erste `WaitWObj`-Instruktion baut eine Verbindung zu dem Objekt im Startfenster auf. Die zweite `WaitWObj` wird sofort beendet, wenn das Objekt noch verbunden

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.337 WaitWObj - Warten auf Werkobjekt am Förderer
Conveyor Tracking
Fortsetzung

ist. Sie wartet jedoch auf das nächste Objekt, wenn das vorherige Objekt den maximalen Abstand überschritten hat oder verworfen wurde.

Beispiel 4:

```
WaitWObj wobj_on_cnv1\RelDist:=500.0\MaxTime:=0.1 \Timeflag:=flag1;
```

Die `WaitWobj`-Instruktion kehrt sofort zurück, wenn das Objekt den Punkt 500 mm überschritten hat. Ist das nicht der Fall, wartet sie 0,1 s auf ein Objekt. Wenn kein Objekt den Punkt 500 mm in diesen 0,1 s passiert, kehrt die Instruktion mit flag1 = `TRUE` zurück.

Einschränkungen

Es dauert 50 ms, bis die Verbindung zum ersten Objekt im Startfenster aufgebaut wird. Sobald eine Verbindung besteht, benötigt die zweite `WaitWObj`-Instruktion mit dem optionalen Argument `\RelDist` nur die normale Abarbeitungszeit für RAPID-Instruktionen.

Syntax

```
WaitWObj
  [ WObj ':='] < persistent (PERS) of wobjdata> ';'
  [ '\' RelDist ':=' < expression (IN) of num > ]
  [ '\' MaxTime ':=' <expression (IN) of num>]
  [ '\' TimeFlag ':=' <variable (VAR) of bool>] ';
```

Weitere Informationen

Information über	Siehe
Ablegen des Werkobjekts am Conveyor	*DropWObj - Ablegen des Werkobjekts am Conveyor auf Seite 193*
Conveyor Tracking	*Anwendungshandbuch - Conveyor Tracking*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.338 WarmStart – Die Steuerung neu starten

Verwendung

`WarmStart` wird zum Neustarten der Steuerung verwendet.

Die Systemparameter können in RAPID mit der Instruktion `WriteCfgData` geändert werden. Sie müssen die Steuerung neu starten, damit eine Änderung Auswirkungen auf einige Systemparameter haben kann. Der Neustart kann mit der Instruktion `WarmStart` erfolgen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WarmStart`:

Beispiel 1:

```
WriteCfgData "/MOC/MOTOR_CALIB/rob1_1","cal_offset",offset1;
WarmStart;
```

Schreibt den Wert der `num` -Variablen `offset1` als Kalibrierungsoffset für Achse 1 in `rob1` und generiert einen Neustart der Steuerung.

Programmabarbeitung

`Warmstart` wird sofort durchgeführt und der Programmzeiger wird auf die nächste Instruktion gesetzt.

Syntax

```
WarmStart ';'
```

Weitere Informationen

Information über	Siehe
Schreibattribut eines Systemparameters	*WriteCfgData - Schreibt ein Attribut eines Systemparameters auf Seite 1123*
Konfiguration	*Technisches Referenzhandbuch - Systemparameter*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.339 WHILE – So lange wiederholen bis...

Verwendung

WHILE wird verwendet, wenn eine Reihe von Instruktionen so lange wiederholt werden soll, wie ein vorgegebener Bedingungsausdruck als „TRUE“ (WAHR) berechnet wird.



Tipp

Wenn es möglich ist, die Anzahl an Wiederholungen zu bestimmen, können Sie die FOR-Instruktion verwenden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion WHILE:

Beispiel 1:

```
WHILE reg1 < reg2 DO
  ...
  reg1 := reg1 + 1;
ENDWHILE
```

Wiederholt die Instruktionen in der WHILE-Schleife, solange reg1 < reg2.

Argumente

```
WHILE Condition DO ... ENDWHILE
```

Condition

Datentyp: bool

Die Bedingung, die den Wert TRUE ergeben muss, damit die Instruktionen in der WHILE-Schleife abgearbeitet werden.

Programmabarbeitung

1. Der Bedingungsausdruck wird berechnet. Wenn der Ausdruck den Wert TRUE ergibt, werden die Instruktionen im WHILE-Block abgearbeitet.
2. Der Bedingungsausdruck wird dann erneut berechnet, und wenn diese Berechnung den Wert TRUE ergibt, werden die Instruktionen im WHILE-Block erneut abgearbeitet.
3. Dieser Prozess wird fortgesetzt, bis das Ergebnis der Ausdrucksberechnung FALSE ist.

Die Wiederholung wird dann beendet und die Programmabarbeitung fährt mit der Instruktion nach der WHILE-Schleife fort.

Wenn das Ergebnis der Ausdrucksberechnung gleich zu Beginn FALSE lautet, werden die Instruktionen im WHILE-Block nicht ausgeführt und das Programm fährt direkt mit der Instruktion nach dem WHILE-Block fort.

Syntax

```
WHILE <conditional expression> DO
  <statement list>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
ENDWHILE
```

Weitere Informationen

Information über	Siehe
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Expressions*
Vorgegebene Anzahl an Wiederholungen	*FOR - Vorgegebene Anzahl an Wiederholungen auf Seite 220*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.340 WorldAccLim - Steuern der Beschleunigung im Welt-Koordinatensystem

Verwendung

`WorldAccLim` (*World Acceleration Limitation*) wird verwendet, um die Beschleunigung/Verzögerung des Werkzeugs (und der Nutzlast) im Welt-Koordinatensystem zu beschränken.

Die Beschränkung wird gleichzeitig im Schwerpunkt des aktuellen Werkzeugs, der aktuellen Nutzlast (falls vorhanden) und im Befestigungsflansch des Roboters erzielt.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WorldAccLim`:

Beispiel 1:

```
WorldAccLim \On := 3.5;
```

Beschleunigung/Verzögerung ist auf 3,5 m/s ^{2} beschränkt.

Beispiel 2:

```
WorldAccLim \Off;
```

Die Beschleunigung/Verzögerung wird auf das Maximum zurückgesetzt (Standard).

Argumente

```
WorldAccLim [\On] | [\Off]
```

`[\On]`

Datentyp: `num`

Der absolute Wert der Beschleunigungs-/Verzögerungsbeschränkung in m/s^{2}.

`[\Off]`

Datentyp: `switch`

Keine Beschränkung. Maximale Beschleunigung (Standard).

Programmabarbeitung

Die Beschleunigungsbeschränkung gilt für die nächste abgearbeitete Instruktion für die Roboterbewegung und ist gültig, bis eine neue `WorldAccLim`-Instruktion abgearbeitet wurde.

Die maximale Beschleunigung (`WorldAccLim \Off`) wird automatisch gesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Es wird empfohlen, nur eine Beschränkungsart für die Beschleunigung zu verwenden. Bei Kombination der Instruktionen `WorldAccLim`, `AccSet` und `PathAccLim` verringert das System die Beschleunigung/Verzögerung in der folgenden Reihenfolge:

- gemäß `WorldAccLim`
- gemäß `AccSet`
- gemäß `PathAccLim`

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ACC_TOO_LOW`	Das Argument `On` ist auf einen zu niedrigen Wert gesetzt.

Einschränkungen

Die minimal zulässige Beschleunigung beträgt 0,1 m/s^2.

Andere Robotermodelle werden nicht unterstützt und können die Instruktion `WorldAccLim` nicht verwenden.

- IRB 340, IRB 360, IRB 540, IRB 1400, IRB 1410

Syntax

```
WorldAccLim
  [ '\' On ':=' <expression (IN) of num>] | ['\' Off ] ';'
```

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Daten für Bewegungseinstellungen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Reduktion der Beschleunigung	*AccSet - Verringern der Beschleunigung auf Seite 25*
Beschränkung der Beschleunigung entlang der Bahn	*PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn auf Seite 507*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.341 Write – In eine zeichenbasierte Datei oder ein E/A-Gerät schreiben

Verwendung

`Write` wird verwendet, um in eine zeichenorientierte Datei oder ein E/A-Gerät zu schreiben. Auch der Wert von bestimmten Daten kann als Text geschrieben werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `Write`:

Beispiel 1:

```
Write logfile, "Execution started";
```

Der Text `Execution started` wird in die Datei mit dem Referenznamen `logfile` geschrieben.

Beispiel 2:

```
VAR num reg1:=5;
...
Write logfile, "No of produced parts="\Num:=reg1;
```

Der Text `No of produced parts=5` wird in die Datei mit dem Referenznamen `logfile` geschrieben.

Argumente

```
Write IODevice String [\Num] | [\Bool] | [\Pos] | [\Orient] |
     [\Dnum] [\NoNewLine]
```

`IODevice`

Datentyp: `iodev`

Name (Verweis) der aktuellen Datei oder des E/A-Geräts.

`String`

Datentyp: `string`

Der zu schreibende Text.

`[\Num]`

Numeric

Datentyp: `num`

Die Daten, deren numerische Werte nach der Zeichenfolge geschrieben werden sollen.

`[\Bool]`

Boolean

Datentyp: `bool`

Die Daten, deren logische Werte nach der Zeichenfolge geschrieben werden sollen.

`[\Pos]`

Position

Datentyp: `pos`

Die Daten, deren Position nach der Zeichenfolge geschrieben werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\Orient]

Orientation

Datentyp: orient

Die Daten, deren Orientierung nach der Zeichenfolge geschrieben werden soll.

[\Dnum]

Numeric

Datentyp: dnum

Die Daten, deren numerische Werte nach der Zeichenfolge geschrieben werden sollen.

[\NoNewLine]

Datentyp: switch

Auslassen des Zeilenvorschubzeichens, das normalerweise das Ende des Texts anzeigt, d. h., die nächste write-Instruktion wird in derselben Zeile fortgesetzt.

Programmabarbeitung

Die Zeichenfolge wird in eine angegebene Datei oder auf ein E/A-Gerät geschrieben. Ein Zeilenvorschubzeichen wird ebenfalls geschrieben. Dies kann jedoch durch Verwendung des Arguments \NoNewLine verhindert werden.

Wenn eines der Argumente \Num, \Bool, \Pos oder \Orient verwendet wird, wird sein Wert zuerst in eine Textzeichenfolge konvertiert, bevor er der ersten Zeichenfolge hinzugefügt wird. Die Konvertierung des Werts in eine Textzeichenfolge findet wie folgt statt:

Argument	Wert	Zeichenfolge
\Num	23	"23"
\Num	1.141367	"1.14137"
\Bool	TRUE	"TRUE"
\Pos	[1817.3,905.17,879.11]	"[1817.3,905.17,879.11]"
\Orient	[0.96593,0,0.25882,0]	"[0.96593,0,0.25882,0]"
\Dnum	4294967295	"4294967295"

Der Wert wird in eine Zeichenfolge mit RAPID-Standardformat konvertiert. Dies bedeutet im Prinzip 6 signifikante Stellen. Wenn der Dezimalteil kleiner als 0,000005 oder größer als 0,999995 ist, wird die Zahl auf eine Ganzzahl gerundet.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ iodev wird zurückgesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.341 Write – In eine zeichenbasierte Datei oder ein E/A-Gerät schreiben

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_FILEACC	Fehler beim Schreiben.

Einschränkungen

Die Argumente \Num, \Dnum, \Bool, \Pos und \Orient schließen einander aus und können daher nicht gemeinsam in derselben Instruktion verwendet werden.

Diese Instruktion kann nur für Dateien oder E/A-Geräte verwendet werden, die zum Schreiben geöffnet wurden.

Syntax

```
Write
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ String ':='] <expression (IN) of string>
  [ '\' Num ':=' <expression (IN) of num> ]
  | ['\' Bool ':=' <expression (IN) of bool> ]
  | ['\' Pos ':=' <expression (IN) of pos> ]
  | ['\' Orient ':=' <expression (IN) of orient> ]
  | ['\' Dnum ':=' <expression (IN) of dnum> ]
  [ '\' NoNewLine] ';'
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ String ':='] <expression (IN) of string>
  [ '\' Num ':=' <expression (IN) of num> ]
  | ['\' Bool ':=' <expression (IN) of bool> ]
  | ['\' Pos ':=' <expression (IN) of pos> ]
  | ['\' Orient ':=' <expression (IN) of orient> ]
  | ['\' Dnum ':=' <expression (IN) of dnum> ]
  [ '\' NoNewLine]
  [ '\' ISOLatin1Encoding] ';'
```

Weitere Informationen

Information über	Siehe
Eine Datei oder ein E/A-Gerät öffnen	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.342 WriteAnyBin - Schreibt Daten in eine zeichenorientierte Datei oder ein zeichenorientiertes E/A-Gerät

Verwendung

`WriteAnyBin` (*Write Any Binary*) wird verwendet, um beliebige Datentypen in eine binäre serielle Datei oder ein binäres E/A-Gerät zu schreiben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteAnyBin`:

Beispiel 1:

```
VAR iodev file1;
VAR orient quat1 := [1, 0, 0, 0];
...
Open "HOME:" \File:= "FILE1.DOC", file1 \Bin;
WriteAnyBin file1, quat1;
```

Die `orient`-Daten `quat1` werden auf den Kanal `channel1` geschrieben.

Argumente

```
WriteAnyBin IODevice Data
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) der Binärdatei oder des binären E/A-Geräts für den Schreibvorgang.

`Data`

Datentyp: `anytype`

Zu schreibende Daten.

Programmabarbeitung

Die erforderliche Anzahl an Bytes wird für die angegebenen Daten in die Binärdatei oder in das binäre E/A-Gerät geschrieben.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Schreiben.

Einschränkungen

Diese Instruktion kann nur für Dateien oder E/A-Geräte verwendet werden, die zum binären Schreiben geöffnet wurden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.342 WriteAnyBin - Schreibt Daten in eine zeichenorientierte Datei oder ein zeichenorientiertes E/A-Gerät

RobotWare - OS

Fortsetzung

Die mit der Instruktion `WriteAnyBin` zu schreibenden Daten müssen von einem Wertdatentyp sein, z. B. `num`, `bool` oder `string`. Ein Datensatz, eine Datensatzkomponente, ein Datenfeld oder eine Datenfeldkomponente dieser Wertdatentypen können ebenfalls verwendet werden. Vollständige Daten oder Teildaten mit einem festen Wert (Semi-Value) oder keinem Wert (Non-Value) können nicht verwendet werden.

Weil `WriteAnyBin-ReadAnyBin` nur interne Steuerungsdaten zwischen Robotersteuerungen sendet, wird kein Datenprotokoll veröffentlicht, und die Daten können auf keinem PC interpretiert werden.

> **Hinweis**
>
> Die Entwicklung von Steuerungssoftware kann dazu führen, dass die Kompatibilität nicht mehr gegeben ist. Daher ist es nicht möglich, `WriteAnyBin-ReadAnyBin` in unterschiedlichen Versionen von RobotWare zu verwenden

Syntax

```
WriteAnyBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Data ':='] <expression (IN) of anytype> ';'
```

Weitere Informationen

Information über	Siehe
Öffnen von E/A-Geräten oder Dateien	*Technisches Referenzhandbuch - RAPID Overview*
Lesen von Daten aus einem binären E/A-Gerät oder einer Binärdatei	*ReadAnyBin – Lesen von Daten aus einem binären E/A-Gerät oder einer Binärdatei auf Seite 581*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.343 WriteBin - Schreibt in ein binäres E/A-Gerät

Verwendung

`WriteBin` wird verwendet, um eine Anzahl von Bytes in ein binäres E/A-Gerät zu schreiben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteBin`:

Beispiel 1:

```
WriteBin channel2, text_buffer, 10;
```

`10` Zeichen aus der Liste `text_buffer` werden auf den Kanal `channel2` geschrieben.

Argumente

```
WriteBin IODevice Buffer NChar
```

`IODevice`

Datentyp: `iodev`

Name (Verweis) des E/A-Geräts.

`Buffer`

Datentyp: `array of num`

Die Liste (das Datenfeld) mit den zu schreibenden Zahlen (Zeichen).

`NChar`

Number of Characters

Datentyp: `num`

Die Anzahl der Zeichen, die aus dem `Buffer` geschrieben werden sollen.

Programmabarbeitung

Die angegebene Anzahl an Zahlen (Zeichen) in der Liste, die auf das E/A-Gerät geschrieben werden sollen.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Schreiben.

Einschränkungen

Diese Instruktion kann nur für E/A-Geräte verwendet werden, die zum binären Schreiben geöffnet wurden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.343 WriteBin - Schreibt in ein binäres E/A-Gerät

RobotWare - OS

Fortsetzung

Syntax

```
WriteBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Buffer ':='] <array {*} (IN) of num> ','
  [ NChar ':='] <expression (IN) of num> ';'
```

Weitere Informationen

Information über	Siehe
E/A-Geräte öffnen etc.	*Technisches Referenzhandbuch - RAPID Overview*
Konvertieren von string-Daten in Byte-Daten	*StrToByte - Konvertieren von string-Daten in Byte-Daten auf Seite 1535*
Byte-Daten	*byte – Ganzzahlige Werte 0-255 auf Seite 1644*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.344 WriteBlock – Datenblock auf Gerät schreiben

Verwendung

`WriteBlock` dient zum Schreiben eines Datenblocks auf ein Gerät, das mit der seriellen Sensorschnittstelle verbunden ist. Die Daten werden aus einer Datei abgerufen.

Die Sensorschnittstelle kommuniziert mithilfe des RTP1-Transportprotokolls mit Sensoren über serielle Kanäle.

Dies ist ein Beispiel für eine Sensorkanalkonfiguration.

```
COM_PHY_CHANNEL:
  Name "COM1:"
  Connector "COM1"
  Baudrate 19200
COM_TRP:
  Name "sen1:"
  Type "RTP1"
  PhyChannel "COM1"
```

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteBlock`:

Beispiel 1:

```
CONST string SensorPar := "flp1:senpar.cfg";
CONST num ParBlock:= 1;

! Connect to the sensor device "sen1:" (defined in sio.cfg).
SenDevice "sen1:";

! Write sensor parameters from flp1:senpar.cfg
! to sensor datablock 1.

WriteBlock "sen1:", ParBlock, SensorPar;
```

Argumente

```
WriteBlock device BlockNo FileName [ \TaskName ]
```

`device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`BlockNo`

Datentyp: `num`

Das Argument `BlockNo` wird verwendet, um den Datenblock im zu schreibenden Sensorblock auszuwählen.

`FileName`

Datentyp: `string`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.344 WriteBlock – Datenblock auf Gerät schreiben

Sensor Interface

Fortsetzung

Das Argument `FileName` wird verwendet, um eine Datei auszuwählen, aus der die Daten in den mithilfe des Arguments `BlockNo` ausgewählten Sensordatenblock gespeichert werden sollen.

`[ \TaskName ]`

Datentyp: `string`

Das Argument `TaskName` ermöglicht, auf Geräte in anderen RAPID-Tasks zuzugreifen.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Fehlerkonstante (ERRNO Wert)	Beschreibung
SEN_NO_MEAS	Messfehler
SEN_NOREADY	Sensor kann Befehl nicht handhaben
SEN_GENERRO	Allgemeiner Sensorfehler
SEN_BUSY	Sensorbus
SEN_UNKNOWN	Unbekannter Sensor
SEN_EXALARM	Externer Sensorfehler
SEN_CAALARM	Interner Sensorfehler
SEN_TEMP	Sensortemperaturfehler
SEN_VALUE	Unzulässiger Kommunikationswert
SEN_CAMCHECK	Sensorprüffehler
SEN_TIMEOUT	Kommunikationsfehler

Syntax

```
WriteBlock
  [ device ':=' ] < expression(IN) of string> ','
  [ BlockNo ':=' ] < expression (IN) of num > ','
  [ FileName ':=' ] < expression (IN) of string > ','
  [ '\' TaskName ':=' < expression (IN) of string > ] ';'
```

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Schreiben einer Sensorvariablen	*WriteVar – Variable schreiben auf Seite 1132*
Lesen eines Sensordatenblocks	*ReadBlock - Lesen eines Datenblocks von einem Gerät auf Seite 584*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.345 WriteCfgData - Schreibt ein Attribut eines Systemparameters

Verwendung

`WriteCfgData` wird verwendet, um ein Attribut eines benannten Systemparameters (Konfigurationsdaten) zu schreiben.

Außer dem Schreiben von benannten Parametern können auch nicht benannte Parameter gesucht und aktualisiert werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Instruktion `WriteCfgData`. In beiden Beispielen wird gezeigt, wie benannte Parameter geschrieben werden.

Beispiel 1:

```
VAR num offset1 := 1.2;
...
WriteCfgData "/MOC/MOTOR_CALIB/rob1_1","cal_offset",offset1;
```

Der in die `num`-Variable `offset1` geschriebene Kalibrierungsoffset für Achse `1` in `rob_1`.

Beispiel 2:

```
VAR string io_device := "my_device";
...
WriteCfgData "/EIO/EIO_SIGNAL/process_error","Device",io_device;
```

Der in die `string`-Variable `io_device`, geschriebene Name des E/A-Geräts, in der das Signal `process_error` definiert ist.

Argumente

```
WriteCfgData InstancePath Attribute CfgData [\ListNo]
```

`InstancePath`

Datentyp: `string`

Gibt den Pfad zu dem Parameter an, auf den zugegriffen werden soll.

Für benannte Parameter ist das Format dieser Zeichenfolge `/DOMAIN/TYPE/ParameterName`.

Für unbenannte Parameter ist das Format dieser Zeichenfolge `/DOMAIN/TYPE/Attribute/AttributeValue`.

`Attribute`

Datentyp: `string`

Der Attributname des zu schreibenden Parameters.

`CfgData`

Datentyp: `anytype`

Das Datenobjekt, aus dem die neuen zu speichernden Daten gelesen werden. Je nach Attributtyp sind folgende Typen zulässig: `bool`, `num`, `dnum` oder `string`.

`[\ListNo]`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.345 WriteCfgData - Schreibt ein Attribut eines Systemparameters
RobotWare - OS
Fortsetzung

Variable, in der die Instanznummer des zu suchenden und zu aktualisierenden `Attribute` + `AttributeValue` gespeichert wird.

Das erste Vorkommen von `Attribute` + `AttributeValue` hat die Instanznummer 0. Wenn mehrere Instanzen gesucht werden, wird der in `\ListNo` zurückgegebene Wert um 1 erhöht. Wenn hingegen keine weiteren Instanzen vorhanden sind, ist der zurückgegebene Wert -1. Mit der vordefinierten Konstanten `END_OF_LIST` kann überprüft werden, ob mehrere zu suchende Instanzen vorhanden sind.

Programmabarbeitung

Der Wert des im Argument `Attribute` angegebenen Attributs wird gemäß dem Wert des im Argument `CfgData` angegebenen Datenobjekts gesetzt.

Bei Verwendung des Formats `/DOMAIN/TYPE/ParameterName` in `InstancePath` kann nur auf benannte Parameter zugegriffen werden, d. h. Parameter, deren erstes Attribut `name`, `Name` oder `NAME` lautet.

Verwenden Sie für unbenannte Parameter den optionalen Parameter `\ListNo`, um anzugeben, in welche Instanz der Attributwert geschrieben werden soll. Er wird nach jedem erfolgreichen Schreiben mit der nächsten verfügbaren Instanz aktualisiert, in die geschrieben werden soll.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_CFG_NOTFND`	In der Konfigurationsdatenbank konnten die Daten mit der Kennzeichnung “`InstancePath` + `Attribute`” nicht gefunden werden.
`ERR_CFG_ILLTYPE`	Der Datentyp für den Parameter `CfgData` ist nicht mit dem echten Datentyp für die gefundenen Daten identisch, die in der Konfigurationsdatenbank mit “`InstancePath` + `Attribute`” angegeben werden.
`ERR_CFG_LIMIT`	Die Daten für Parameter `CfgData` überschreiten bestimmte Grenzen (max./min. Wert).
`ERR_CFG_INTERNAL`	Versuch, interne schreibgeschützte Daten zu schreiben.
`ERR_CFG_OUTOFBOUNDS`	Der Wert der Variablen im Argument `\ListNo` liegt bei der Abarbeitung der Instruktion außerhalb des Bereichs verfügbarer Instanzen (0 ... n).

Weitere Beispiele

Weitere Beispiele für die Instruktion `WriteCfgdata` sind unten aufgeführt. In beiden Beispielen wird gezeigt, wie unbenannte Parameter geschrieben werden.

Beispiel 1:

```
VAR num read_index;
VAR num write_index;
VAR string read_str;
...
read_index:=0;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
write_index:=0;
ReadCfgData "/EIO/EIO_CROSS/Act1/do_13", "Res", read_str,
     \ListNo:=read_index;
WriteCfgData "/EIO/EIO_CROSS/Act1/do_13", "Res", "my"+read_str,
     \ListNo:=write_index;
```

Liest das resultierende Signal für das unbenannte digitale Aktorensignal `do_13` und fügt den Namen in die Zeichenfolgenvariable `read_str` ein. Anschließend wird der Name auf `di_13` mit dem Präfix „`my`“ aktualisiert.

In diesem Beispiel weist die Domäne EIO folgenden Konfigurationscode auf:

EIO_CROSS:

-Name "Cross_di_1_do_2" -Res "di_1" -Act1 "do_2"

-Name "Cross_di_2_do_2" -Res "di_2" -Act1 "do_2"

-Name "Cross_di_13_do_13" -Res "di_13" -Act1 "do_13"

Beispiel 2:

```
VAR num read_index;
VAR num write_index;
VAR string read_str;
...
read_index:=0;
write_index:=0;
WHILE read_index <> END_OF_LIST DO
  ReadCfgData "/EIO/EIO_SIGNAL/Device/USERIO", "Name", read_str,
       \ListNo:=read_index;
  IF read_index <> END_OF_LIST THEN
    WriteCfgData "/EIO/EIO_SIGNAL/Device/USERIO", "Name",
         "my"+read_str, \ListNo:=write_index;
  ENDIF
ENDWHILE
```

Liest die Namen aller für die E/A-Gerät `USERIO` definierten Signale. Ändert die Namen der Signale in den gelesenen Namen mit dem Präfix "`my`".

In diesem Beispiel weist die Domäne EIO folgenden Konfigurationscode auf:

```
EIO_SIGNAL:
  -Name "USERDO1" -SignalType "DO" -Device "USERIO" -DeviceMap "0"
  -Name "USERDO2" -SignalType "DO" -Device "USERIO" -DeviceMap "1"
  -Name "USERDO3" -SignalType "DO" -Device "USERIO" -DeviceMap "2"
```

Einschränkungen

Die Umrechnung der RAPID-Programmeinheiten (mm, Grad, Sekunden usw.) in Systemparametereinheiten (m, rad, Sekunden usw.) für `CfgData` des Datentyps `num` und `dnum` muss vom Anwender im Rapid-Programm durchgeführt werden.

Bei den meisten Systemparametern müssen Sie die Steuerung manuell neu starten oder die Instruktion `WarmStart` ausführen, damit die Änderung wirksam wird. Systemparameter, die ohne Neustart aus RobotStudio oder FlexPendant geändert werden können, erfordern ebenfalls keinen Neustart, wenn sie aus RAPID geändert werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.345 WriteCfgData - Schreibt ein Attribut eines Systemparameters

RobotWare - OS

Fortsetzung

Bei Verwendung des Formats `/DOMAIN/TYPE/ParameterName` in `InstancePath` kann nur auf benannte Parameter zugegriffen werden, d. h. Parameter, deren erstes Attribut `name`, `Name` oder `NAME` lautet.

Die Anzahl der Zeichen in RAPID-Zeichenfolgen ist auf 80 beschränkt. In einigen Fällen kann das theoretisch für die Definition von `InstancePath`, `Attribute` oder `CfgData` zu wenig sein.

Vordefinierte Daten

Mithilfe der vordefinierten Konstanten `END_OF_LIST` mit dem Wert -1 kann der Schreibvorgang beendet werden, wenn keine weiteren Instanzen gefunden werden.

Syntax

```
WriteCfgData
  [ InstancePath ':=' ] < expression (IN) of string > ','
  [ Attribute ':=' ] < expression (IN) of string > ','
  [ CfgData ':=' ] < expression (IN) of anytype >
  [ '\' ListNo ':=' < variable (VAR) of num >] ';'
```

Weitere Informationen

Information über	Siehe
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Leseattribut eines Systemparameters	*ReadCfgData - Lesen von Attributen eines Systemparameters auf Seite 586*
Abrufen des Namens der aktuellen Task	*RobName - Abrufen des TCP-Roboternamens auf Seite 1481*
Konfiguration	*Technisches Referenzhandbuch - Systemparameter*
Neustart des Systems	*WarmStart – Die Steuerung neu starten auf Seite 1109*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.346 WriteRawBytes - Schreiben von rawbytes-Daten

Verwendung

`WriteRawBytes` wird verwendet, um Daten vom Typ `rawbytes` von einem mit `Open\Bin` geöffneten Gerät zu schreiben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteRawBytes`:

Beispiel 1:

```
VAR iodev io_device;
VAR rawbytes raw_data_out;
VAR rawbytes raw_data_in;
VAR num float := 0.2;
VAR string answer;

ClearRawBytes raw_data_out;
PackDNHeader "10", "20 1D 24 01 30 64", raw_data_out;
PackRawBytes float, raw_data_out, (RawBytesLen(raw_data_out)+1)
     \Float4;

Open "/FCI1:/dsqc328_1", io_device \Bin;
WriteRawBytes io_device, raw_data_out;
ReadRawBytes io_device, raw_data_in \Time:=1;
Close io_device;
UnpackRawBytes raw_data_in, 1, answer \ASCII:=10;
```

In diesem Beispiel wird `raw_data_out` gelöscht und dann mit einem DeviceNet-Header und dem Float-Wert 0.2 gepackt.

Ein Gerät `"/FCI1/:dsqc328_1"` wird geöffnet und die aktuellen gültigen Daten in `raw_data_out` werden auf das Gerät geschrieben. Dann wartet das Programm höchstens 1 Sekunde auf das Lesen von dem Gerät, das in `raw_data_in` gespeichert ist.

Nachdem das Gerät „`/FCI1/:dsqc328_1`“ geschlossen wurde, werden die gelesenen Daten als Zeichenfolge von `10` Zeichen entpackt und in der Variablen answer gespeichert.

Argumente

```
WriteRawBytes IODevice RawData [\NoOfBytes]
```

`IODevice`

Datentyp: `iodev`

`IODevice` ist der Bezeichner des Geräts, auf das `RawData` geschrieben werden soll.

`RawData`

Datentyp: `rawbytes`

`RawData` ist der Datencontainer, der in `IODevice` geschrieben werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.346 WriteRawBytes - Schreiben von rawbytes-Daten

RobotWare - OS

Fortsetzung

`[\NoOfBytes]`

Datentyp: `num`

`\NoOfBytes` gibt an, wie viele Bytes `RawData` mit Beginn bei Index 1 in `IODevice` geschrieben werden sollen.

Wenn `\NoOfBytes` nicht vorhanden ist, wird die aktuelle Länge gültiger Bytes in der Variablen `RawData` in das Gerät IODevice geschrieben.

Programmabarbeitung

Bei der Programmabarbeitung werden Daten in das mit `IODevice` angegebene Gerät geschrieben.

Wird `WriteRawBytes` für Feldbusbefehle wie DeviceNet verwendet, sendet der Feldbus immer eine Antwort. Die Antwort muss in RAPID mit der `ReadRawBytes`-Instruktion behandelt werden.

Die aktuelle Länge der gültigen Bytes in der `RawData`-Variablen wird nicht verändert.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Schreiben.

Syntax

```
WriteRawBytes
  [ IODevice ':=' ] < variable (VAR) of iodev> ','
  [ RawData ':=' ] < variable (VAR) of rawbytes>
  [ '\' NoOfBytes ':=' < expression (IN) of num>] ';'
```

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.347 WriteStrBin - Schreibt einer Zeichenfolge auf ein binäres E/A-Gerät

Verwendung

`WriteStrBin` (*Write String Binary*) wird verwendet, um eine Zeichenfolge in ein binäres E/A-Gerät oder eine binäre Datei zu schreiben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteStrBin`:

Beispiel 1:

```
WriteStrBin channel2, "Hello World\0A";
```

Der String `"Hello World\0A"` wird in den von `channel2` referenzierten Kanal geschrieben. Die Zeichenfolge wird in diesem Fall mit einer neuen Zeile `\\0A` beendet. Alle Zeichen und hexadezimalen Werte, die mit `WriteStrBin` geschrieben werden, bleiben vom System unverändert.

Argumente

```
WriteStrBin IODevice Str
```

`IODevice`

Datentyp: `iodev`

Name (Verweis) des E/A-Geräts.

`Str`

String

Datentyp: `string`

Der zu schreibende Text.

Programmabarbeitung

Die Zeichenfolge wird auf das angegebenen E/A-Gerät oder die Datei geschrieben.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Schreiben.

Einschränkungen

Diese Instruktion kann nur für E/A-Geräte oder Dateien verwendet werden, die zum binären Lesen und Schreiben geöffnet wurden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.347 WriteStrBin - Schreibt einer Zeichenfolge auf ein binäres E/A-Gerät

RobotWare - OS

Fortsetzung

Syntax

```
WriteStrBin
  [ IODevice ':='] <variable (VAR) of iodev> ','
  [ Str ':='] <expression (IN) of string> ';'
```

Weitere Informationen

Information über	Siehe
E/A-Geräte öffnen etc.	*Technisches Referenzhandbuch - RAPID Overview*
Read binary string	*ReadStrBin - Liest Zeichenfolge von binären E/A-Gerät oder einer Datei auf Seite 1472*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.348 WriteVar – Variable schreiben

Verwendung

`WriteVar` wird für das Schreiben einer Variablen auf ein Gerät verwendet, das mit der seriellen Sensorschnittstelle verbunden ist.

Die Sensorschnittstelle kommuniziert mit Sensoren über E/A-Geräte.

Konfigurationsbeispiel

Dies ist ein Beispiel für eine Sensorkanalkonfiguration.

Diese Parameter gehören zum Typ *Transmission Protocol* in der Parametergruppe *Communication*.

Name	Type	Remote Address	Remote Port
sen1:	SOCKDEV	192.168.125.101	6344

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteVar`:

Beispiel 1:

```
! Define variable numbers
CONST num SensorOn := 6;
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;
VAR pos SensorPos;

! Connect to the sensor device“ sen1:” (defined in sio.cfg).
SenDevice "sen1:";

! Request start of sensor meassurements
WriteVar "sen1:", SensorOn, 1;

! Read a cartesian position from the sensor.
SensorPos.x := ReadVar "sen1:", XCoord;
SensorPos.y := ReadVar "sen1:", YCoord;
SensorPos.z := ReadVar "sen1:", ZCoord;

! Stop sensor
WriteVar "sen1:", SensorOn, 0;
```

Argumente

```
WriteVar device VarNo VarData [ \TaskName ]
```

`device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`VarNo`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Mit dem Argument `VarNo` wird die Sensorvariable ausgewählt.

`VarData`

Datentyp: `num`

Das Argument `VarData` definiert die Daten, die in die Variable geschrieben werden sollen, die durch das Argument `VarNo` ausgewählt wurde.

`[ \TaskName ]`

Datentyp: `string`

Das Argument `TaskName` ermöglicht, auf Geräte in anderen RAPID-Tasks zuzugreifen.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Fehlerkonstante (ERRNO-Wert)	Beschreibung
`SEN_NO_MEAS`	Messfehler
`SEN_NOREADY`	Sensor kann Befehl nicht handhaben
`SEN_GENERRO`	Allgemeiner Sensorfehler
`SEN_BUSY`	Sensorbus
`SEN_UNKNOWN`	Unbekannter Sensor
`SEN_EXALARM`	Externer Sensorfehler
`SEN_CAALARM`	Interner Sensorfehler
`SEN_TEMP`	Sensortemperaturfehler
`SEN_VALUE`	Unzulässiger Kommunikationswert
`SEN_CAMCHECK`	Sensorprüffehler
`SEN_TIMEOUT`	Kommunikationsfehler

Syntax

```
WriteVar
  [device ':='] <expression (IN) of string> ','
  [VarNo ':='] <expression (IN) of num> ','
  [VarData ':='] <expression (IN) of num> ','
  ['\' TaskName ':=' <expression (IN) of string>] ';'
```

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Lesen einer Sensorvariablen	*ReadVar - Lesen einer Variablen von einem Gerät auf Seite 1474*
Schreiben eines Sensordatenblocks	*WriteBlock – Datenblock auf Gerät schreiben auf Seite 1121*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.348 WriteVar – Variable schreiben

Sensor Interface

Fortsetzung

Information über	Siehe
Lesen eines Sensordatenblocks	*ReadBlock - Lesen eines Datenblocks von einem Gerät auf Seite 584*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.349 WriteVarArr - Schreiben mehrerer Variablen auf ein Sensorgerät

Verwendung

`WriteVarArr` wird verwendet, um bis zu sechs Variablen gleichzeitig auf ein Sensorgerät zu schreiben.

Der Sensor muss über die RobotWare-Option *Sensor Interface* konfiguriert werden und kommunizieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WriteVarArr`:

Beispiel 1:

```
! Define variable numbers
CONST num jointno := 16;
CONST num unit := 19;
VAR sensorvardata writeData{3};

! Connect to the sensor device “sen1:” (defined in sio.cfg).
SenDevice "sen1:";

! Setup two variables to write
writeData{1}:=[jointno, 0, false, 1, 5];
writeData{2}:=[unit, 0, false, 1, 1];
! A varNumber of -1 will be ignored
writeData{3}:=[-1, 0, false, 1, 1];

WriteVarArr "sen1:", writeData;
```

Das Beispiel zeigt eine Schreibanforderung der Variablen `jointno` und `unit`.

Argumente

```
WriteVarArr Device, Data, [\taskName]
```

`Device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`Data`

Datentyp: `sensorvardata`

Eine Datenfeldvariable, die eine Datendefinition der Variablen bezeichnet, die geschrieben werden.

`[ \TaskName ]`

Datentyp: `string`

Das Argument `TaskName` ermöglicht, auf Geräte in anderen RAPID-Tasks zuzugreifen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.349 WriteVarArr - Schreiben mehrerer Variablen auf ein Sensorgerät

Sensor Interface

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
SEN_NO_MEAS	Messfehler
SEN_NOREADY	Sensor kann Befehl nicht handhaben
SEN_GENERRO	Allgemeiner Sensorfehler
SEN_BUSY	Sensor belegt
SEN_UNKNOWN	Unbekannter Sensor
SEN_EXALARM	Externer Sensorfehler
SEN_CAALARM	Interner Sensorfehler
SEN_TEMP	Sensortemperaturfehler
SEN_VALUE	Unzulässiger Kommunikationswert
SEN_CAMCHECK	Sensorprüffehler
SEN_TIMEOUT	Kommunikationsfehler

Syntax

```
WriteVarArr
  [Device ':='] <expression(IN) of string> ','
  [Data ':=' ] < array variable {*} (INOUT) of sensorvardata > ','
  ['\' TaskName ':=' <expression (IN) of string> ] ';'
```

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Mehrere Variablen von einem Gerät ablesen	*ReadVarArr - Mehrere Variablen von einem Sensorgerät ablesen auf Seite 596*
Schreiben einer Sensorvariablen	*WriteVar – Variable schreiben auf Seite 1132*
Lesen einer Sensorvariablen	*ReadVar - Lesen einer Variablen von einem Gerät auf Seite 1474*
Schreiben eines Sensordatenblocks	*WriteBlock – Datenblock auf Gerät schreiben auf Seite 1121*
Lesen eines Sensordatenblocks	*ReadBlock - Lesen eines Datenblocks von einem Gerät auf Seite 584*
Mehrere variable Konfigurationsdaten für die Sensorschnittstelle	*sensorvardata - Mehrere variable Konfigurationsdaten für die Sensorschnittstelle auf Seite 1785*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.350 WZBoxDef - Definieren einer quaderförmigen Weltzone

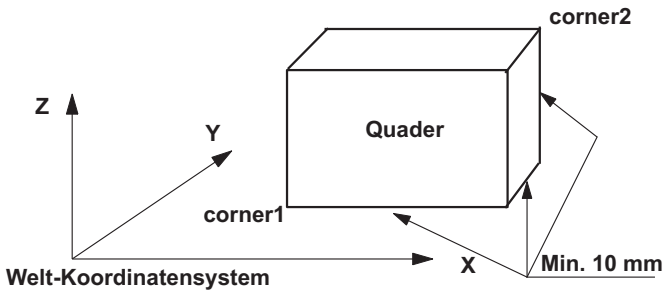
Verwendung

`WZBoxDef` (*World Zone Box Definition*) wird verwendet, um eine Weltzone in der Form eines Quaders zu definieren, dessen Seiten parallel zu den Achsen des Welt-Koordinatensystems verlaufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZBoxDef`:

Beispiel 1:



xx0500002205

```
VAR shapedata volume;
CONST pos corner1:=[200,100,100];
CONST pos corner2:=[600,400,400];
...
WZBoxDef \Inside, volume, corner1, corner2;
```

Definiert ein Quader mit Koordinaten, die parallel zu den Achsen des Welt-Koordinatensystems verlaufen und durch die gegenüberliegenden Ecken `corner1` und `corner2` definiert sind.

Argumente

```
WZBoxDef [\Inside] | [\Outside] Shape LowPoint HighPoint
```

`[\Inside]`

Datentyp: `switch`

Definiert das Volumen innerhalb des Quaders.

`[\Outside]`

Datentyp: `switch`

Definiert das Volumen außerhalb des Quaders (inverses Volumen).

Eines der Argumente `\Inside` oder `\Outside` muss angegeben werden.

`Shape`

Datentyp: `shapedata`

Variable zur Speicherung des definierten Volumens (systeminterne Daten).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.350 WZBoxDef - Definieren einer quaderförmigen Weltzone

World Zones
Fortsetzung

`LowPoint`

Datentyp: `pos`

Position (x,y,z) in mm zur Definition einer unteren Ecke des Quaders.

`HighPoint`

Datentyp: `pos`

Position (x,y,x) in mm zur Definition einer diagonal gegenüberliegenden Ecke des Quaders.

Programmabarbeitung

Die Definition des Quaders wird in einer Variablen des Typs `shapedata` (Argument `Shape`) zur späteren Verwendung in den Instruktionen `WZLimSup` oder `WZDOSet` gespeichert.

Einschränkungen

Die Positionen `LowPoint` und `HighPoint` müssen gegenüberliegende Ecken angeben (mit unterschiedlichem x-, y- und z-Koordinatenwert).

Wenn mit dem Roboter `LowPoint` oder `HighPoint` bestimmt werden soll, muss `wobj0` aktiv sein (Verwendung der Komponente `trans` in `robtarget` z. B. `p1.trans` als Argument).

Syntax

```
WZBoxDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ LowPoint ':='] <expression (IN) of pos> ','
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ HighPoint ':='] <expression (IN) of pos> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Definieren einer Weltzone für Grenzachsen	*WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung auf Seite 1157*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.351 WZCylDef - Definieren einer zylinderförmigen Weltzone

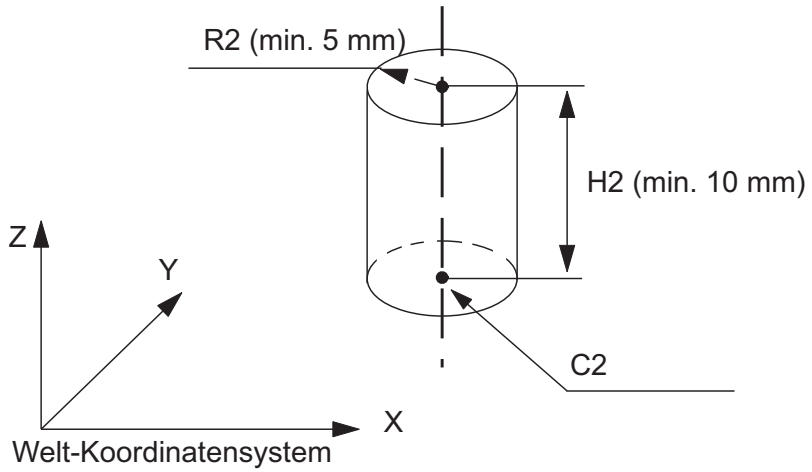
Verwendung

`WZCylDef` (*World Zone Cylinder Definition*) wird verwendet, um eine Weltzone in der Form eines Zylinders zu definieren, dessen Achse parallel zur z-Achse des Welt-Koordinatensystems verläuft.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZCylDef`:

Beispiel 1:



xx0500002206

```
VAR shapedata volume;
CONST pos C2:=[300,200,200];
CONST num R2:=100;
CONST num H2:=200;
...
WZCylDef \Inside, volume, C2, R2, H2;
```

Definiert einen Zylinder mit dem Mittelpunkt des unteren Kreises in `C2`, Radius `R2` und Höhe `H2`.

Argumente

```
WZCylDef [\Inside] | [\Outside] Shape CentrePoint Radius Height
```

`[\Inside]`

Datentyp: `switch`

Definiert das Volumen innerhalb des Zylinders.

`[\Outside]`

Datentyp: `switch`

Definiert das Volumen außerhalb des Zylinders (inverses Volumen).

Eines der Argumente `\Inside` oder `\Outside` muss angegeben werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.351 WZCylDef - Definieren einer zylinderförmigen Weltzone

World Zones
Fortsetzung

`Shape`

Datentyp: `shapedata`

Variable zur Speicherung des definierten Volumens (systeminterne Daten).

`CentrePoint`

Datentyp: `pos`

Position (x,y,x) in mm zur Definition des Mittelpunkts von einem kreisförmigen Abschluss des Zylinders.

`Radius`

Datentyp: `num`

Der Radius des Zylinders in mm.

`Height`

Datentyp: `num`

Die Höhe des Zylinders in mm. Wenn sie positiv ist (+z-Richtung), ist das Argument `CentrePoint` der Mittelpunkt des unteren Zylinderendes (wie im obigen Beispiel). Wenn sie negativ ist (-z-Richtung), ist das Argument `CentrePoint` der Mittelpunkt des oberen Zylinderendes.

Programmabarbeitung

Die Definition des Zylinders wird in der Variablen vom Typ `shapedata` gespeichert (Argument `Shape`), zur späteren Verwendung in der Instruktion `WZLimSup` oder `WZDOSet`.

Einschränkungen

Wenn mit dem Roboter der `CentrePoint` bestimmt werden soll, muss das Werkobjekt `wobj0` aktiv sein (Verwendung der Komponente `trans` in `robtarget`, z. B. `p1.trans` als Argument).

Syntax

```
WZCylDef
  [ '\' Inside] | [ '\' Outside] ','
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ CentrePoint ':='] <expression (IN) of pos> ','
  [ Radius ':='] <expression (IN) of num> ','
  [ Height ':='] <expression (IN) of num> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.351 WZCylDef - Definieren einer zylinderförmigen Weltzone

World Zones

Fortsetzung

Information über	Siehe
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Definieren einer Weltzone für Grenzachsen	*WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung auf Seite 1157*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.352 WZDisable - Deaktivieren der temporären Weltzonenüberwachung

Verwendung

`WZDisable` (*World Zone Disable*) wird verwendet, um die Überwachung einer temporären Weltzone zu deaktivieren, die zuvor definiert wurde, um die Bewegung zu stoppen oder einen Ausgang zu setzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZDisable`:

Beispiel 1:

```
VAR wztemporary wzone;
...
PROC...
  WZLimSup \Temp, wzone, volume;
  MoveL p_pick, v500, z40, tool1;
  WZDisable wzone;
  MoveL p_place, v200, z30, tool1;
ENDPROC
```

Bei der Bewegung zu `p_pick` wird die Position des Roboter-TCPs überwacht, damit er nicht in das angegebene Volumen `wzone` fahren kann. Es erfolgt keine Überwachung, wenn sich der Roboter zu `p_place` bewegt.

Argumente

```
WZDisable WorldZone
```

`WorldZone`

Datentyp: `wztemporary`

Variable oder persistente Variable des Typs `wztemporary`, die die Identität der zu deaktivierenden Weltzone enthält.

Programmabarbeitung

Die temporäre Weltzone wird aktiviert. Das bedeutet, dass die Überwachung des TCP in Relation zum entsprechenden Volumen temporär gestoppt wird. Sie kann über die Instruktion `WZEnable` wieder aktiviert werden.

Einschränkungen

Nur eine temporäre Weltzone kann deaktiviert werden. Eine stationäre Weltzone ist immer aktiv.

Syntax

```
WZDisable
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.352 WZDisable - Deaktivieren der temporären Weltzonenüberwachung

World Zones

Fortsetzung

Information über	Siehe
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Temporäre Weltzonendaten	*wztemporary - Temporäre Weltzonendaten auf Seite 1858*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*
Aktivieren einer Weltzone	*WZEnable - Aktivieren einer temporären Weltzonenüberwachung auf Seite 1149*
Löschen einer Weltzone	*WZFree - Löschen der temporären Weltzonenüberwachung auf Seite 1151*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.353 WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs

Verwendung

`WZDOSet` (*World Zone Digital Output Set*) wird verwendet, um eine Weltzone zur Überwachung der Roboterbewegungen zu aktivieren und die zugehörige Aktion zu definieren..

Wenn sich der TCP des Roboters oder der Roboter/die externen Achsen vor der Abarbeitung dieser Instruktion in der definierten Weltzone oder in der Nähe befinden, wird ein digitales Ausgangssignal auf den angegebenen Wert gesetzt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZDOSet`:

Siehe auch *Weitere Beispiele auf Seite 1146*.

Beispiel 1:

```
VAR wztemporary service;

PROC zone_output()
  VAR shapedata volume;
  CONST pos p_service:=[500,500,700];
  ...
  WZSphDef \Inside, volume, p_service, 50;
  WZDOSet \Temp, service \Inside, volume, do_service, 1;
ENDPROC
```

Die kugelförmige temporäre Weltzone `service` wird im Anwendungsprogramm definiert und das Signal `do_service` wird gesetzt, sobald sich der TCP des Roboters bei der Programmabarbeitung oder beim manuellen Bewegen innerhalb der definierten Kugel befindet.

Argumente

```
WZDOSet [\Temp] | [\Stat] WorldZone [\Inside] | [\Before] Shape
     Signal SetValue
```

`[\Temp]`

Temporary

Datentyp: `switch`

Die zu definierende Weltzone ist eine temporäre Weltzone.

`[\Stat]`

Stationary

Datentyp: `switch`

Die zu definierende Weltzone ist eine stationäre Weltzone.

Eines der Argumente `\Temp` oder `\Stat` muss angegeben werden.

`WorldZone`

Datentyp: `wztemporary` oder `wzstationary`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Variable oder persistente Variable, die mit der Identität (numerischer Wert) der Weltzone aktualisiert wird.

Wenn der Schalter `\Temp` verwendet wird, muss der Datentyp `wztemporary` lauten. Wenn der Schalter `\Stat` verwendet wird, muss der Datentyp `wzstationary` lauten.

`[\Inside]`

Datentyp: `switch`

Das digitale Ausgangssignal wird gesetzt, wenn sich der TCP oder festgelegte Achsen des Roboters innerhalb des definierten Volumens befinden.

`[\Before]`

Datentyp: `switch`

Das digitale Ausgangssignal wird gesetzt, bevor der TCP oder festgelegte Achsen des Roboters das definierte Volumen erreichen (so bald wie möglich vor dem Volumen).

Eines der Argumente `\Inside` oder `\Before` muss angegeben werden.

`Shape`

Datentyp: `shapedata`

Die Variable, die das Volumen der Weltzone definiert.

`Signal`

Datentyp: `signaldo`

Der Name des digitalen Ausgangssignals, das geändert wird.

Für eine stationäre Weltzone muss das Signal die richtige *interne* Zugriffsebene haben. Die Zugriffsebene wird in der Signaldefinition in den E/A-Systemparametern festgelegt. Diese Konzepte werden in *Technisches Referenzhandbuch - Systemparameter* weiter ausgeführt, siehe Typen *Signal* und *Access Level*. Die Zugriffsebene muss für den Zugriff durch den Benutzer (RAPID, FlexPendant) geschützt sein, damit die vordefinierte interne Zugriffsebene verwendet oder eine benutzerdefinierte Zugriffsebene vom Benutzer definiert werden kann.

`SetValue`

Datentyp: `dionum`

Gewünschter Wert des Signals (0 oder 1), wenn sich der TCP des Roboters im Volumen oder kurz vor dem Eintritt in das Volumen befindet.

Außerhalb oder gerade noch außerhalb des Volumens wird das Signal auf den entgegengesetzten Wert gesetzt.

Programmabarbeitung

Die definierte Weltzone wird aktiviert. Ab diesem Moment wird die Position des TCP (oder des Roboters/der externen Achsen) überwacht und der Ausgang wird gesetzt, wenn sich diese Position innerhalb des Volumens (`\Inside`) oder knapp vor der Volumengrenze (`\Before`) befindet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.353 WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs

World Zones

Fortsetzung

Bei der Verwendung von `WZHomeJointDef` oder `WZLimJointDef` gemeinsam mit `WZDOSet` wird das digitale Ausgangssignal nur gesetzt, wenn sich alle aktiven Achsen mit Achsenraumüberwachung vor dem oder im Achsenraum befinden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `WZDOSet` sind unten aufgeführt.

Beispiel 1:

```
VAR wztemporary home;
VAR wztemporary service;
PERS wztemporary equip1:=[0];

PROC main()
  ...
  ! Definition of all temporary world zones
  zone_output;
  ...
  ! equip1 in robot work area
  WZEnable equip1;
  ...
  ! equip1 out of robot work area
  WZDisable equip1;
  ...
  ! No use for equip1 any more
  WZFree equip1;
  ...
ENDPROC

PROC zone_output()
  VAR shapedata volume;
  CONST pos p_home:=[800,0,800];
  CONST pos p_service:=[800,800,800];
  CONST pos p_equip1:=[-800,-800,0];
  ...
  WZSphDef \Inside, volume, p_home, 50;
  WZDOSet \Temp, home \Inside, volume, do_home, 1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.353 WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs

World Zones

Fortsetzung

```
  WZSphDef \Inside, volume, p_service, 50;
  WZDOSet \Temp, service \Inside, volume, do_service, 1;
  WZCylDef \Inside, volume, p_equip1, 300, 1000;
  WZLimSup \Temp, equip1, volume;
  ! equip1 not in robot work area
  WZDisable equip1;
ENDPROC
```

Definition der temporären Weltzonen `home` und `service` im Anwendungsprogramm, um die Signale `do_home` und `do_service` zu setzen, wenn sich der Roboter während der Programmabarbeitung oder manuellen Bewegung in der Kugel `home` bzw. `service` befindet.

Zusätzlich wird die temporäre Weltzone `equip1` definiert, die nur in dem Programmteil aktiv ist, in dem sich `equip1` innerhalb des Arbeitsbereichs des Roboters befindet. Zu diesem Zeitpunkt stoppt der Roboter, bevor er in das Volumen `equip1` eintritt, sowohl bei der Programmabarbeitung als auch beim manuellen Bewegen. `equip1` kann von anderen Programmtasks aus mithilfe des Werts der persistenten Variablen `equip1` deaktiviert oder aktiviert werden.

Einschränkungen

Eine Weltzone kann nicht mit derselben Variable im Argument `WorldZone` neu definiert werden.

Eine stationäre Weltzone kann im RAPID-Programm nicht deaktiviert, erneut aktiviert oder gelöscht werden.

Eine temporäre Weltzone kann im RAPID-Programm deaktiviert (`WZDisable`), erneut aktiviert (`WZEnable`) oder gelöscht (`WZFree`) werden.

Syntax

```
WZDOSet
  [['\' Temp] | ['\' Stat] ',']
  [WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
  ['\' Inside] | ['\' Before] ','
  [Shape ':='] <variable (VAR) of shapedata> ','
  [Signal ':='] <variable (VAR) of signaldo> ','
  [SetValue ':='] <expression (IN) of dionum> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Temporäre Weltzone	*wztemporary - Temporäre Weltzonendaten auf Seite 1858*
Stationäre Weltzone	*wzstationary - Stationäre Weltzonendaten auf Seite 1856*
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.353 WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs

World Zones

Fortsetzung

Information über	Siehe
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Signalzugriffsebene	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.354 WZEnable - Aktivieren einer temporären Weltzonenüberwachung

Verwendung

`WZEnable` (*World Zone Enable*) wird verwendet, um die Überwachung einer temporären Weltzone erneut zu aktivieren, die zuvor definiert wurde, um die Bewegung zu stoppen oder einen Ausgang zu setzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZEnable`:

Beispiel 1:

```
VAR wztemporary wzone;
...
PROC ...
  WZLimSup \Temp, wzone, volume;
  MoveL p_pick, v500, z40, tool1;
  WZDisable wzone;
  MoveL p_place, v200, z30, tool1;
  WZEnable wzone;
  MoveL p_home, v200, z30, tool1;
ENDPROC
```

Bei der Bewegung zu `p_pick` wird die Position des Roboter-TCPs überwacht, damit er nicht in das angegebene Volumen `wzone` fahren kann. Es erfolgt keine Überwachung, wenn sich der Roboter zu `p_place` bewegt. Aber die Überwachung wird erneut aktiviert, bevor er sich zu `p_home` bewegt.

Argumente

```
WZEnable WorldZone
```

`WorldZone`

Datentyp: `wztemporary`

Variable oder persistente Variable des Typs `wztemporary`, die die Identität der zu aktivierenden Weltzone enthält.

Programmabarbeitung

Die temporäre Weltzone wird erneut aktiviert. Beachten Sie, dass eine Weltzone bei ihrer Erstellung automatisch aktiviert wird. Sie muss nur erneut aktiviert werden, wenn sie zuvor durch `WZDisable` deaktiviert wurde.

Einschränkungen

Nur eine temporäre Weltzone kann deaktiviert und erneut aktiviert werden. Eine stationäre Weltzone ist immer aktiv.

Syntax

```
WZEnable
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.354 WZEnable - Aktivieren einer temporären Weltzonenüberwachung

World Zones

Fortsetzung

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Temporäre Weltzonendaten	*wztemporary - Temporäre Weltzonendaten auf Seite 1858*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*
Deaktivieren einer Weltzone	*WZDisable - Deaktivieren der temporären Weltzonenüberwachung auf Seite 1142*
Löschen einer Weltzone	*WZFree - Löschen der temporären Weltzonenüberwachung auf Seite 1151*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.355 WZFree - Löschen der temporären Weltzonenüberwachung

Verwendung

`WZFree` (*World Zone Free*) wird verwendet, um die Definition einer temporären Weltzone zu löschen, die zuvor definiert wurde, um die Bewegung zu stoppen oder einen Ausgang zu setzen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZFree`:

Beispiel 1:

```
VAR wztemporary wzone;
...
PROC ...
  WZLimSup \Temp, wzone, volume;
  MoveL p_pick, v500, z40, tool1;
  WZDisable wzone;
  MoveL p_place, v200, z30, tool1;
  WZEnable wzone;
  MoveL p_home, v200, z30, tool1;
  WZFree wzone;
ENDPROC
```

Bei der Bewegung zu `p_pick` wird die Position des Roboter-TCPs überwacht, damit er nicht in das angegebene Volumen `wzone` fahren kann. Es erfolgt keine Überwachung, wenn sich der Roboter zu `p_place` bewegt. Aber die Überwachung wird erneut aktiviert, bevor er sich zu `p_home` bewegt. Wenn diese Position erreicht ist, wird die Weltzonendefinition gelöscht.

Argumente

```
WZFree WorldZone
```

`WorldZone`

Datentyp: `wztemporary`

Variable oder persistente Variable des Typs `wztemporary`, die die Identität der zu löschenden Weltzone enthält.

Programmabarbeitung

Die temporäre Weltzone wird deaktiviert, bevor ihre Definition gelöscht wird.

Eine einmal gelöschte Weltzone lässt sich nicht mehr aktivieren oder deaktivieren.

Einschränkungen

Nur eine temporäre Weltzone kann deaktiviert, erneut aktiviert oder gelöscht werden. Eine stationäre Weltzone ist immer aktiv.

Syntax

```
WZFree
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ';'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.355 WZFree - Löschen der temporären Weltzonenüberwachung

World Zones

Fortsetzung

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Temporäre Weltzonendaten	*wztemporary - Temporäre Weltzonendaten auf Seite 1858*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*
Deaktivieren einer Weltzone	*WZDisable - Deaktivieren der temporären Weltzonenüberwachung auf Seite 1142*
Aktivieren einer Weltzone	*WZEnable - Aktivieren einer temporären Weltzonenüberwachung auf Seite 1149*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.356 WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen

Verwendung

`WZHomeJointDef` (*World Zone Home Joint Definition*) wird verwendet, um eine Weltzone in Achsenkoordinaten für den Roboter und externe Achsen zu definieren, die als Grund- oder Wartungsposition verwendet werden soll.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZHomeJointDef`:

Beispiel 1:

```
VAR wzstationary home;
...
PROC power_on()
  VAR shapedata joint_space;
  CONST jointtarget home_pos := [ [ 0, 0, 0, 0, 0, -45], [ 0, 9E9,
       9E9, 9E9, 9E9, 9E9] ];
  CONST jointtarget delta_pos := [ [ 2, 2, 2, 2, 2, 2], [ 5, 9E9,
       9E9, 9E9, 9E9, 9E9] ];
  ...
  WZHomeJointDef \Inside, joint_space, home_pos, delta_pos;
  WZDOSet \Stat, home \Inside, joint_space, do_home, 1;
ENDPROC
```

Definition und Aktivierung der stationären Weltzone `home`, die das Signal `do_home to 1`, setzt, wenn sich alle Roboterachsen und die externe Achse `extax.eax_a` während der Programmabarbeitung und Bewegung an der Achsenposition `home_pos` befinden (innerhalb von +/- `delta_pos` für jede Achse). Die Variable `joint_space` vom Datentyp `shapedata` wird verwendet, um Daten von der Instruktion `WZHomeJointDef` in die Instruktion `WZDOSet` zu übertragen.

Argumente

```
WZHomeJointDef [\Inside] | [\Outside] Shape MiddleJointVal
     DeltaJointVal
```

`[\Inside]`

Datentyp: `switch`

Definiert den Achsenraum innerhalb von `MiddleJointVal` +/- `DeltaJointVal`.

`[\Outside]`

Datentyp: `switch`

Definiert den Achsenraum außerhalb von `MiddleJointVal` +/- `DeltaJointVal` (inverser Achsenraum).

`Shape`

Datentyp: `shapedata`

Variable zur Speicherung des definierten Achsenraums (systeminterne Daten).

`MiddleJointVal`

Datentyp: `jointtarget`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.356 WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen
World Zones
Fortsetzung

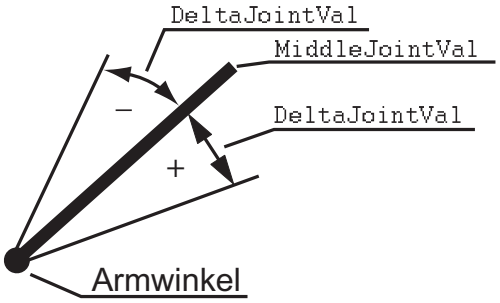
Die Position der Achsenkoordinaten für die Mitte des zu definierenden Achsenraums. Wird für jede Roboterachse und externe Achse angegeben (Grad für Rotationsachsen und mm für lineare Achsen). Wird in absoluten Achsen angegeben (nicht im Offset-Koordinatensystem `EOffsSet-EOffsOn` für externe Achsen). **Wert 9E9 für eine Achse bedeutet, dass die Achse nicht überwacht werden soll.** Eine nicht aktive externe Achse liefert zum Zeitpunkt der Programmierung ebenfalls 9E9.

`DeltaJointVal`

Datentyp: `jointtarget`

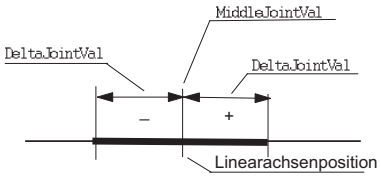
Die +/- delta-Position der Achsenkoordinaten von der Mitte des Achsenraums. Der Wert muss für alle zu überwachenden Achsen größer 0 sein.

Die folgende Abbildung zeigt die Definition des Achsenraums für eine Rotationsachse.



xx0500002208

Die folgende Abbildung zeigt die Definition des Achsenraums für eine lineare Achse.



xx0500002209

Programmabarbeitung

Die Definition des Achsenraums wird in der Variablen vom Typ `shapedata` gespeichert (Argument `Shape`), zur späteren Verwendung in der Instruktion `WZLimSup` oder `WZDOSet`.

Bei der Verwendung von `WZHomeJointDef` gemeinsam mit `WZDOSet` wird das digitale Ausgangssignal nur gesetzt, wenn sich alle aktiven Achsen mit Achsenraumüberwachung vor dem oder im Achsenraum befinden.

Bei Verwendung von `WZHomeJointDef` mit Achsenraum außerhalb (Argument `\Outside`) zusammen mit `WZLimSup` wird der Roboter gestoppt, sobald eine überwachte aktive Achse den Achsenraum erreicht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.356 WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen

World Zones
Fortsetzung

Bei Verwendung von `WZHomeJointDef` mit Achsenraum innerhalb (Argument `\Inside`) zusammen mit `WZLimSup` wird der Roboter gestoppt, sobald die letzte überwachte aktive Achse den Achsenraum erreicht. Das bedeutet, dass eine oder mehrere Achsen, aber nicht alle aktiven und überwachten Achsen gleichzeitig innerhalb des Arbeitsraums sein können.

Beim Abarbeiten der Instruktion `ActUnit` oder `DeactUnit` zur Aktivierung oder Deaktivierung von mechanischen Einheiten wird der Überwachungsstatus für die HOME-Position oder Arbeitsbereichsbeschränkung aktualisiert.

Einschränkungen



xx0100000002

Nur aktive mechanische Einheiten und ihre bei der Aktivierung der Weltzone aktiven Achsen (mit Instruktion `WZDOSet` bzw. `WZLimSup`) werden bei der Überwachung der HOME-Position bzw. der Beschränkung des Arbeitsbereichs berücksichtigt. Zusätzlich müssen die mechanische Einheit und ihre Achsen bei der Bewegung durch das Programm oder beim manuellen Bewegen aktiv sein, damit sie überwacht werden können.

Wenn sich z. B. eine überwachte Achse außerhalb ihrer HOME-Achsenposition befindet, aber deaktiviert ist, wird nicht verhindert, dass das digitale Ausgangssignal für die HOME-Achsenposition gesetzt wird, falls sich alle übrigen überwachten aktiven Achsen innerhalb der HOME-Achsenposition befinden. Beim erneuten Aktivieren dieser Achse wird sie wieder überwacht, das Robotersystem befindet sich außerhalb der HOME-Achsenposition und der digitale Ausgang wird zurückgesetzt.

Syntax

```
WZHomeJointDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ MiddleJointVal ':='] <expression (IN) of jointtarget> ','
  [ DeltaJointVal ':='] <expression (IN) of jointtarget> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.356 WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen

World Zones

Fortsetzung

Information über	Siehe
Definieren einer Weltzone für Grenzachsen	*WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung auf Seite 1157*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.357 WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung

Verwendung

`WZLimJointDef` (*World Zone Limit Joint Definition*) wird verwendet, um eine Weltzone in Achsenkoordinaten für den Roboter und externe Achsen zu definieren, die für die Beschränkung des Arbeitsbereichs verwendet werden sollen.

Mit `WZLimJointDef` ist es möglich, den Arbeitsraum für jeden Roboter und externe Achsen im RAPID-Programm zu begrenzen, und zwar unabhängig von der Begrenzung, die mit den Systemparametern *Motion - Arm - robx_y - Upper Joint Bound ... Lower Joint Bound* durchgeführt werden kann.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZLimJointDef`:

Beispiel 1:

```
VAR wzstationary work_limit;
  ...
  PROC power_on()
    VAR shapedata joint_space;
    CONST jointtarget low_pos:= [ [ -90, 9E9, 9E9, 9E9, 9E9, 9E9],
         [ -1000, 9E9, 9E9, 9E9, 9E9, 9E9]];
    CONST jointtarget high_pos := [ [ 90, 9E9, 9E9, 9E9,9E9, 9E9],
         [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9] ];
    ...
    WZLimJointDef \Outside, joint_space, low_pos, high_pos;
    WZLimSup \Stat, work_limit, joint_space;
  ENDPROC
```

Definition und Aktivierung der stationären Weltzone `work_limit`, die bei der Programmabarbeitung und dem manuellen Bewegen den Arbeitsraum für die Roboterachse 1 auf -90 und +90 Grad und die externe Achse `extax.eax_a` auf -1.000 mm einschränkt. Die Variable `joint_space` vom Datentyp `shapedata` wird verwendet, um Daten von der Instruktion `WZLimJointDef` in die Instruktion `WZLimSup` zu übertragen.

Argumente

```
WZLimJointDef [\Inside] | [\Outside] Shape LowJointVal HighJointVal
```

`[\Inside]`

Datentyp: `switch`

Definiert den Achsenraum innerhalb von `LowJointVal ... HighJointVal`.

`[\Outside]`

Datentyp: `switch`

Definiert den Achsenraum außerhalb von `LowJointVal ... HighJointVal`(inverser Achsenraum).

`Shape`

Datentyp: `shapedata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.357 WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung
World Zones
Fortsetzung

Variable zur Speicherung des definierten Achsenraums (systeminterne Daten).

`LowJointVal`

Datentyp: `jointtarget`

Die Position der Achsenkoordinaten für die Untergrenze des zu definierenden Achsenraums. Wird für jede Roboterachse und externe Achse angegeben (Grad für Rotationsachsen und mm für lineare Achsen). Wird in absoluten Achsen angegeben (nicht im Offset-Koordinatensystem `EOffsSet` oder `EOffsOn` für externe Achsen). Wert `9E9` für eine Achse bedeutet, dass die Untergrenze für die Achse nicht überwacht werden soll. Eine nicht aktive externe Achse liefert zum Zeitpunkt der Programmierung ebenfalls `9E9`.

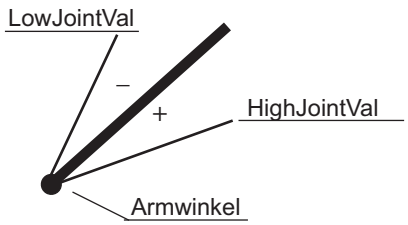
`HighJointVal`

Datentyp: `jointtarget`

Die Position der Achsenkoordinaten für die Obergrenze des zu definierenden Achsenraums. Wird für jede Roboterachse und externe Achse angegeben (Grad für Rotationsachsen und mm für lineare Achsen). Wird in absoluten Achsen angegeben (nicht im Offset-Koordinatensystem `EOffsSet` oder `EOffsOn` für externe Achsen). Wert `9E9` für eine Achse bedeutet, dass die Obergrenze für die Achse nicht überwacht werden soll. Eine nicht aktive externe Achse liefert zum Zeitpunkt der Programmierung ebenfalls `9E9`.

`HighJointVal` minus `LowJointVal` muss für jede der zu überwachenden Achsen größer 0 sein.

Die folgende Abbildung zeigt die Definition des Achsenraums für eine Rotationsachse.



xx0500002281

Die folgende Abbildung zeigt die Definition des Achsenraums für eine lineare Achse.



xx0100000002

Programmabarbeitung

Die Definition des Achsenraums wird in der Variablen vom Typ `shapedata` gespeichert (Argument `Shape`), zur späteren Verwendung in der Instruktion `WZLimSup` oder `WZDOSet`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Bei der Verwendung von `WZLimJointDef` gemeinsam mit `WZDOSet` wird das digitale Ausgangssignal nur gesetzt, wenn sich alle aktiven Achsen mit Achsenraumüberwachung vor dem oder im Achsenraum befinden.

Bei Verwendung von `WZLimJointDef` mit Achsenraum außerhalb (Argument `\Outside`) zusammen mit `WZLimSup` wird der Roboter gestoppt, sobald eine überwachte aktive Achse den Achsenraum erreicht.

Bei Verwendung von `WZLimJointDef` mit Achsenraum innerhalb (Argument `\Inside`) zusammen mit `WZLimSup` wird der Roboter gestoppt, sobald die letzte überwachte aktive Achse den Achsenraum erreicht. Das bedeutet, dass eine oder mehrere Achsen, aber nicht alle aktiven und überwachten Achsen gleichzeitig innerhalb des Arbeitsraums sein können.

Beim Abarbeiten der Instruktion `ActUnit` oder `DeactUnit` wird der Überwachungsstatus aktualisiert.

Einschränkungen



xx0100000002

WARNUNG!

Nur aktive mechanische Einheiten und ihre bei der Aktivierung der Weltzone aktiven Achsen (mit Instruktion `WZDOSet` bzw. `WZLimSup`) werden bei der Überwachung der HOME-Position bzw. der Beschränkung des Arbeitsbereichs berücksichtigt. Zusätzlich müssen die mechanische Einheit und ihre Achsen bei der Bewegung durch das Programm oder beim manuellen Bewegen aktiv sein, damit sie überwacht werden können.

Wenn sich z. B. eine überwachte Achse außerhalb ihrer HOME-Achsenposition befindet, aber deaktiviert ist, wird nicht verhindert, dass das digitale Ausgangssignal für die HOME-Achsenposition gesetzt wird, falls sich alle übrigen überwachten aktiven Achsen innerhalb der HOME-Achsenposition befinden. Beim erneuten Aktivieren dieser Achse wird sie wieder überwacht, das Robotersystem befindet sich außerhalb der Achsen-Grundstellung und der digitale Ausgang wird zurückgesetzt.

Syntax

```
WZLimJointDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ Shape ':='] <variable (VAR) of shapedata> ','
  [ LowJointVal ':='] <expression (IN) of jointtarget> ','
  [ HighJointVal ':='] <expression (IN) of jointtarget> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.357 WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung

World Zones

Fortsetzung

Information über	Siehe
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.358 WZLimSup - Aktivieren der Überwachung der Weltzonengrenze

Verwendung

`WZLimSup` (*World Zone Limit Supervision*) wird verwendet, um die Aktion zu definieren und eine Weltzone zur Überwachung des Arbeitsbereichs des Roboters oder der externen Achsen zu aktivieren.

Wenn der TCP des Roboters oder die Roboter- oder externen Achsen nach Abarbeitung dieser Instruktion die Weltzone erreichen, wird die Bewegung sowohl bei der Programmabarbeitung als auch beim manuellen Bewegen gestoppt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZLimSup`:

Siehe auch *Weitere Beispiele auf Seite 1162*.

Beispiel 1:

```
VAR wzstationary max_workarea;
...
PROC POWER_ON()
  VAR shapedata volume;
  ...
  WZBoxDef \Outside, volume, corner1, corner2;
  WZLimSup \Stat, max_workarea, volume;
ENDPROC
```

Definition und Aktivierung der stationären Weltzone `max_workarea` mit der Form des Bereichs außerhalb eines Quaders (temporär in `volume` gespeichert) und der Überwachung des Aktionsbereichs. Der Roboter stoppt mit einer Fehlermeldung, bevor er in den Bereich außerhalb des Quaders eintritt.

Argumente

```
WZLimSup [\Temp] | [\Stat] WorldZone Shape
```

`[\Temp]`

Temporary

Datentyp: `switch`

Die zu definierende Weltzone ist eine temporäre Weltzone.

`[\Stat]`

Stationary

Datentyp: `switch`

Die zu definierende Weltzone ist eine stationäre Weltzone.

Eines der Argumente `\Temp` oder `\Stat` muss angegeben werden.

`WorldZone`

Datentyp: `wztemporary` oder `wzstationary`

Variable oder persistente Variable, die mit der Identität (numerischer Wert) der Weltzone aktualisiert wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.358 WZLimSup - Aktivieren der Überwachung der Weltzonengrenze
World Zones
Fortsetzung

Wenn der Schalter \Temp verwendet wird, muss der Datentyp wztemporary lauten. Wenn der Schalter \Stat verwendet wird, muss der Datentyp wzstationary lauten.

Shape

Datentyp: shapedata

Die Variable, die das Volumen der Weltzone definiert.

Programmabarbeitung

Die definierte Weltzone wird aktiviert. Ab diesem Moment werden die Positionen des Roboter-TCP bzw. der Roboter-/externen Achsen überwacht. Bei Erreichen des definierten Bereichs wird die Bewegung gestoppt.

Bei Verwendung von WZLimJointDef oder WZHomeJointDef mit Achsenraum außerhalb (Argument \Outside) zusammen mit WZLimSup wird der Roboter gestoppt, sobald eine überwachte aktive Achse den Achsenraum erreicht.

Bei Verwendung von WZLimJointDef oder WZHomeJointDefmit Achsenraum innerhalb (Argument \Inside) zusammen mit WZLimSup wird der Roboter gestoppt, sobald die letzte überwachte aktive Achse den Achsenraum erreicht. Das bedeutet, dass eine oder mehrere Achsen, aber nicht alle aktiven und überwachten Achsen gleichzeitig innerhalb des Arbeitsraums sein können.

Beim Abarbeiten der Instruktion ActUnit oder DeactUnit wird der Überwachungsstatus aktualisiert.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion WZLimSup sind unten aufgeführt.

Beispiel 1:

```
VAR wzstationary box1_invers;
VAR wzstationary box2;

PROC wzone_power_on()
  VAR shapedata volume;
  CONST pos box1_c1:=[500,-500,0];
  CONST pos box1_c2:=[-500,500,500];
  CONST pos box2_c1:=[500,-500,0];
  CONST pos box2_c2:=[200,-200,300];
  ...
  WZBoxDef \Outside, volume, box1_c1, box1_c2;
  WZLimSup \Stat, box1_invers, volume;
  WZBoxDef \Inside, volume, box2_c1, box2_c2;
  WZLimSup \Stat, box2, volume;
ENDPROC
```

Einschränkung des Arbeitsbereichs für den Roboter mit den folgenden stationären Weltzonen:

- Außerhalb des Arbeitsbereichs, wenn außerhalb von box1_invers
- Außerhalb des Arbeitsbereichs, wenn innerhalb von box2

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn diese Routine mit dem Systemereignis POWER ON verbunden ist, sind diese Weltzonen immer im System aktiv, sowohl für Programmbewegung als auch für manuelles Bewegen.

Einschränkungen

Eine Weltzone kann nicht mit derselben Variable im Argument `WorldZone` neu definiert werden.

Eine stationäre Weltzone kann im RAPID-Programm nicht deaktiviert, erneut aktiviert oder gelöscht werden.

Eine temporäre Weltzone kann im RAPID-Programm deaktiviert (`WZDisable`), erneut aktiviert (`WZEnable`) oder gelöscht (`WZFree`) werden.

Syntax

```
WZLimSup
  [ ['\' Temp] | ['\Stat] ',']
  [ WorldZone ':='] <variable or persistent (INOUT) of wztemporary>
       ','
  [ Shape ':='] <variable (VAR) of shapedata> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Temporäre Weltzone	*wztemporary - Temporäre Weltzonendaten auf Seite 1858*
Stationäre Weltzone	*wzstationary - Stationäre Weltzonendaten auf Seite 1856*
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Definieren einer Weltzone für Grenzachsen	*WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung auf Seite 1157*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1.359 WZSphDef - Definieren einer kugelförmigen Weltzone

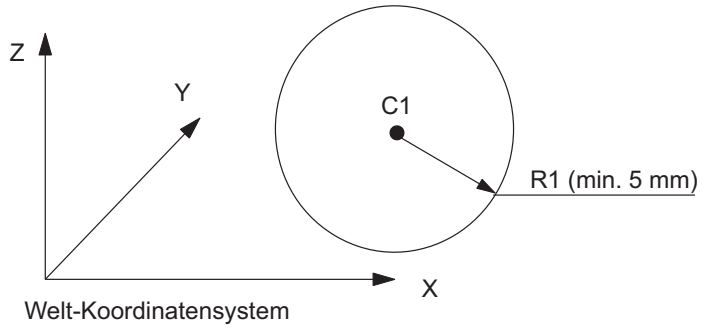
Verwendung

`WZSphDef` (*World Zone Sphere Definition*) wird verwendet, um eine Weltzone in der Form einer Kugel zu definieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Instruktion `WZSphDef`:

Beispiel 1:



xx0500002207

```
VAR shapedata volume;
CONST pos C1:=[300,300,200];
CONST num R1:=200;
...
WZSphDef \Inside, volume, C1, R1;
```

Definiert eine Kugel mit dem Namen `volume` durch ihren Mittelpunkt `C1` und ihren Radius `R1`.

Argumente

```
WZSphDef [\Inside] | [\Outside] Shape CentrePoint Radius
```

`[\Inside]`

Datentyp: `switch`

Definiert das Volumen innerhalb der Kugel.

`[\Outside]`

Datentyp: `switch`

Definiert das Volumen außerhalb der Kugel (inverses Volumen).

Eines der Argumente `\Inside` oder `\Outside` muss angegeben werden.

`Shape`

Datentyp: `shapedata`

Variable zur Speicherung des definierten Volumens (systeminterne Daten).

`CentrePoint`

Datentyp: `pos`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Position (x,y,x) in mm zur Definition des Mittelpunkts der Kugel.

`Radius`

Datentyp: `num`

Der Radius der Kugel in mm.

Programmabarbeitung

Die Definition der Kugel wird in einer Variable des Typs `shapedata` (Argument `Shape`) zur späteren Verwendung in den Instruktionen `WZLimSup` oder `WZDOSet` gespeichert.

Einschränkungen

Wenn mit dem Roboter der `CentrePoint` bestimmt werden soll, muss das Werkobjekt `wobj0` aktiv sein (Verwendung der Komponente `trans` in `robtarget`, z. B. `p1.trans` als Argument).

Syntax

```
WZSphDef
  [ ['\' Inside] | ['\' Outside] ',']
  [ Shape ':='<variable (VAR) of shapedata> ','
  [ CentrePoint ':='] <expression (IN) of pos> ','
  [ Radius ':='] <expression (IN) of num> ';'
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Definieren einer Weltzone für Grenzachsen	*WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung auf Seite 1157*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Diese Seite wurde absichtlich leer gelassen

2 Funktionen

2.1 Abs - Berechnen des Absolutwerts

Verwendung

`Abs` wird verwendet, um den Absolutwert, d. h. einen positiven Wert von numerischen Daten, zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Abs`.

Siehe auch *Weitere Beispiele auf Seite 1167*.

Beispiel 1:

```
reg1 := Abs(reg2);
```

`Reg1` wird der Absolutwert von `reg2` zugewiesen.

Rückgabewert

Datentyp: `num`

Der absolute Wert, also ein positiver numerischer Wert, zum Beispiel:

Eingangswert	Rückgabewert
3	3
-3	3
-2.53	2.53

Argumente

```
Abs (Value)
```

`Value`

Datentyp: `num`

Der eingegebene Wert.

Weitere Beispiele

Weitere Beispiele für die Funktion `Abs` sind unten aufgeführt.

Beispiel 1:

```
TPReadNum no_of_parts, "How many parts should be produced? ";
no_of_parts := Abs(no_of_parts);
```

Der Bediener wird aufgefordert, die Anzahl der zu produzierenden Teile einzugeben. Um sicherzustellen, dass der Wert größer als null ist, wird der vom Bediener eingegebene Wert mit einem Pluszeichen versehen.

Syntax

```
Abs '('
  [ Value ':=' ] < expression (IN) of num >')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.1 Abs - Berechnen des Absolutwerts

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.2 AbsDnum - Ruft den absoluten Wert eines dnum ab

Verwendung

`AbsDnum` wird verwendet, um den Absolutwert, d.h. einen positiven Wert von `dnum` numerischen Daten zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `AbsDnum`.

Siehe auch *Weitere Beispiele auf Seite 1169*.

Beispiel 1:

```
VAR dnum value1;
VAR dnum value2:=-20000000;
value1 := AbsDnum(value2);
```

`Value1` wird der Absolutwert von `value2` zugewiesen.

Rückgabewert

Datentyp: `dnum`

Der absolute Wert, also ein positiver numerischer Wert, zum Beispiel:

Eingangswert	Rückgabewert
3	3
-3	-3
-2.53	2.53
-4503599627370496	4503599627370496

Argumente

```
AbsDnum (Value)
```

`Value`

Datentyp: `dnum`

Der eingegebene Wert.

Weitere Beispiele

Weitere Beispiele für die Funktion `AbsDnum` sind unten aufgeführt.

Beispiel 1:

```
TPReadDnum no_of_parts, "How many parts should be produced? ";
no_of_parts := AbsDnum(no_of_parts);
```

Der Bediener wird aufgefordert, die Anzahl der zu produzierenden Teile einzugeben. Um sicherzustellen, dass der Wert größer als null ist, wird der vom Bediener eingegebene Wert mit einem Pluszeichen versehen.

Syntax

```
AbsDnum '('
  [ Value ':=' ] < expression (IN) of dnum > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.2 AbsDnum - Ruft den absoluten Wert eines dnum ab

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.3 ACos - Berechnen des Arcuscosinus

Verwendung

ACos (*Arc Cosine*) wird zur Berechnung des Arcuscosinuswertes für die Datentypen num verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion ACos.

Beispiel 1:

```
VAR num angle;
VAR num value;
...
...
angle := ACos(value);
```

angle ruft den Arcuscosinus-Wert von value ab.

Rückgabewert

Datentyp: num

Der Arcuscosinuswert in Grad, Bereich [0, 180].

Argumente

```
ACos (Value)
```

Value

Datentyp: num

Der Argumentwert muss im Bereich [-1, 1] liegen.

Einschränkungen

Die Abarbeitung der Funktion Acos(x) ergibt einen Fehler, wenn x außerhalb des Bereichs [-1, 1] liegt.

Syntax

```
ACos '('
  [Value ':='] <expression (IN) of num>')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.4 ACosDnum - Berechnen des Arcuscosinus

Verwendung

ACosDnum (*Arc Cosine dnum*) wird zur Berechnung des Arcuscosinuswertes für die Datentypen dnum verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion ACosDnum.

Beispiel 1:

```
VAR dnum angle;
VAR dnum value;
...
...
angle := ACosDnum(value);
```

angle ruft den Arcuscosinus-Wert von value ab.

Rückgabewert

Datentyp: dnum

Der Arcuscosinuswert in Grad, Bereich [0, 180].

Argumente

```
ACosDnum (Value)
```

Value

Datentyp: dnum

Der Argumentwert muss im Bereich [-1, 1] liegen.

Einschränkungen

Die Abarbeitung der Funktion AcosDnum(x) ergibt einen Fehler, wenn x außerhalb des Bereichs [-1, 1] liegt.

Syntax

```
ACosDnum '('
  [Value ':='] <expression (IN) of dnum>')'
```

Eine Funktion mit einem Rückgabewert des Typs dnum.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.5 AInput - Lesen des Wertes eines analogen Eingangssignals

Verwendung

`AInput` wird verwendet, um den aktuellen Wert eines analogen Eingangssignals zu lesen.

> **Hinweis**
>
> Beachten Sie, dass die Funktion `AInput` eine alte Funktion ist, die nicht länger verwendet werden darf. Beachten Sie die Beispiele für eine alternative und empfohlene Art der Programmierung.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `AInput`.

Siehe auch *Weitere Beispiele auf Seite 1174*.

Beispiel 1:

```
IF AInput(ai1) < 1.5 THEN ...
...
IF ai1 < 1.5 THEN ...
```

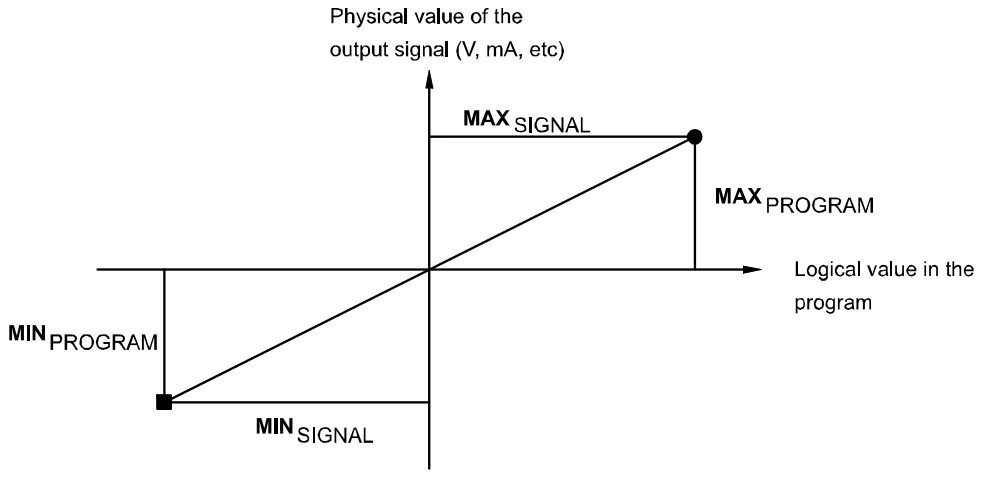
Wenn der aktuelle Wert des Signals `ai1` kleiner als 1,5 ist, dann...

Rückgabewert

Datentyp: `num`

Der aktuelle Wert des Signals.

Der aktuelle Wert wird skaliert (in Übereinstimmung mit den Systemparametern), bevor er vom RAPID-Programm gelesen wird. In der folgenden Abbildung wird ein Diagramm über die Skalierung von analogen Signalwerten gezeigt:



xx0500002408

Argumente

```
AInput (Signal)
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.5 AInput - Lesen des Wertes eines analogen Eingangssignals

RobotWare - OS

Fortsetzung

`Signal`

Datentyp: `signalai`

Der Name des analogen Eingangs, der gelesen werden soll.

Weitere Beispiele

Weitere Beispiele für die Verwendung der Funktion `AInput` sind unten aufgeführt.

Beispiel 1:

```
WHILE AInput(current) > 35 DO ...
...
WHILE current > 35 DO ...
```

Wenn der aktuelle Wert des Signals `current` größer als 35 ist, führen Sie aus...

Beispiel 2:

```
deviation := 3 * AInput(sensor) + 10;
...
deviation := 3 * sensor + 10;
```

Die Abweichung wird auf Grundlage des Wertes des Signals `sensor` berechnet und in der Variablen `deviation` gespeichert.

Syntax

```
AInput '('
  [ Signal ':=' ] < variable (VAR) of signalai > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.6 AND – Bewertet einen logischen Wert

Verwendung

AND ist ein Funktion, die zur Bewertung von zwei bedingten Ausdrücken (true/false) verwendet wird.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion AND.

Beispiel 1:

```
VAR num a;
VAR num b;
VAR bool c;
...
c := a>5 AND b=3;
```

Der Rückgabewert von c ist TRUE, wenn a größer als 5 und b gleich 3 ist. Ansonsten ist der Rückgabewert FALSE.

Beispiel 2:

```
VAR num mynum;
VAR string mystring;
VAR bool mybool;
VAR bool result;
...
result := mystring="Hello" AND mynum<15 OR mybool;
```

Der Rückgabewert von result ist TRUE, wenn mystring sowohl "Hello" ist als auch mynum kleiner als 15 ist. Oder wenn mybool TRUE ist. Ansonsten ist der Rückgabewert FALSE.

Die AND-Anweisung wird zuerst bewertet, danach die OR-Anweisung. Dies wird durch die Klammern in der Zeile unten angezeigt.

```
result := (mystring="Hello" AND mynum<15) OR mybool;
```

Rückgabewert

Datentyp: bool

Der Rückgabewert ist TRUE, wenn beide bedingten Ausdrücke richtig sind; ansonsten ist der Rückgabewert FALSE.

Syntax

```
<expression of bool> AND <expression of bool>
```

Eine Funktion mit einem Rückgabewert des Typs bool.

Weitere Informationen

Information über	Siehe
Logische bitweise AND-Operation mit byte-Daten	*BitAnd - Logische bitweise AND-Operation mit Byte-Daten auf Seite 1189*
Logische bitweise AND-Operation mit dnum-Daten	*BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten auf Seite 1191*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.6 AND – Bewertet einen logischen Wert

RobotWare - OS

Fortsetzung

Information über	Siehe
OR	*OR – Bewertet einen logischen Wert auf Seite 1415*
XOR	*XOR – Bewertet einen logischen Wert auf Seite 1634*
NOT	*NOT – Kehrt einen logischen Wert um auf Seite 1406*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.7 AOutput - Lesen des Werts eines analogen Ausgangssignals

Verwendung

`AOutput` wird verwendet, um den aktuellen Wert eines analogen Ausgangssignals zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `AOutput`.

Beispiel 1:

```
IF AOutput(ao4) > 5 THEN ...
```

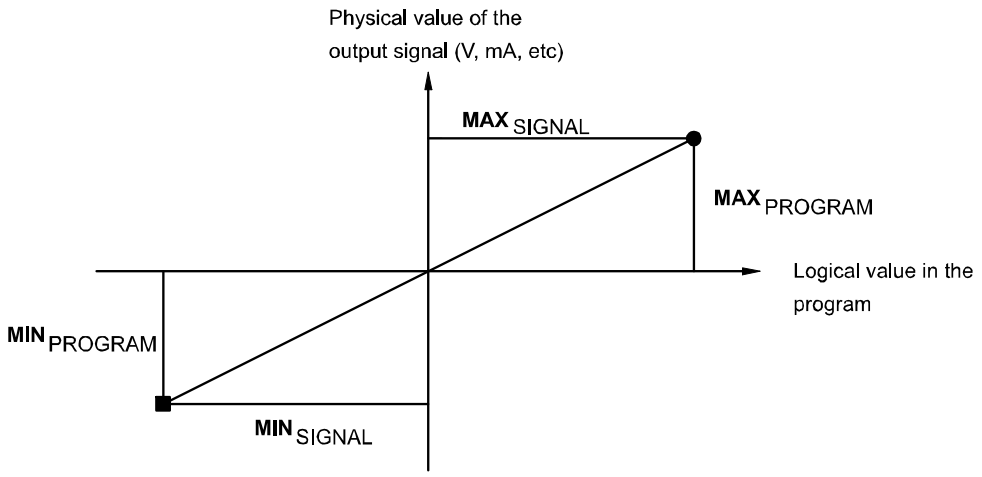
Wenn der aktuelle Wert des Signals `ao4` größer als `5` ist, dann `...`

Rückgabewert

Datentyp: `num`

Der aktuelle Wert des Signals.

Der aktuelle Wert wird skaliert (in Übereinstimmung mit den Systemparametern), bevor er vom RAPID-Programm gelesen wird. In der folgenden Abbildung wird ein Diagramm über die Skalierung von analogen Signalwerten gezeigt:



xx0500002408

Argumente

```
AOutput (Signal)
```

`Signal`

Datentyp: `signalao`

Der Name des analogen Ausgangssignals, das gelesen werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.7 AOutput - Lesen des Werts eines analogen Ausgangssignals
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
AOutput '('
  [ Signal ':=' ] < variable (VAR) of signalao > ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Setzen eines analogen Ausgangssignals	*SetAO - Ändern des Werts eines analogen Ausgangssignals auf Seite 700*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.8 ArgName - Abrufen des Argumentnamens

Verwendung

`ArgName` (*Argument Name*) ruft den Namen des originalen Datenobjekts für das aktuelle Argument oder die aktuellen Daten ab.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ArgName`.

Siehe auch *Weitere Beispiele auf Seite 1180*.

Beispiel 1:

```
VAR num chales :=5;
...
proc1 chales;
PROC proc1 (num par1)
  VAR string name;
  ...
  name:=ArgName(par1);
  TPWrite "Argument name "+name+" with value "\Num:=par1;
ENDPROC
```

Der Variablen `name` wird der Zeichenfolgenwert „`chales`“ zugewiesen und auf FlexPendant wird die folgende Zeichenfolge geschrieben: "Argument name chales with value 5".

Rückgabewert

Datentyp: `string`

Der Name des originalen Datenobjekts.

Argumente

```
ArgName (Parameter [\ErrorNumber])
```

`Parameter`

Datentyp: `anytype`

Der formale Parameterbezeichner (für die Routine, in der sich `ArgName` befindet) oder die Datenidentität.

Es kann ein beliebiger Typ von Daten verwendet werden, z. B. Einzeldaten, Datensätze, Datensatzkomponenten, Datenfelder oder Datenfeldkomponenten.

`ErrorNumber`

Datentyp: `errnum`

Eine Variable (vor ihrer Verwendung wird sie vom System auf 0 gesetzt), die den Fehlercode enthält, wenn das Argument ein Ausdruck ist, nicht vorhanden ist oder den Typ switch hat. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Die Funktion gibt den originalen Namen des Datenobjekts für ein ganzes Objekt des Typs Konstante, Variable oder Persistente zurück. Das originale Datenobjekt kann global oder lokal im Programmmodul oder lokal in einer Routine sein (normale RAPID-Regeln für den Gültigkeitsbereich).

Wenn es Teil eines Datenobjekts ist, wird der Name des ganzen Datenobjekts zurückgegeben.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_ARGNAME	• Argument ist Ausdruckswert. • Argument ist nicht vorhanden. • Argument hat Typ „switch“.

Weitere Beispiele

Weitere Beispiele für die Funktion `ArgName` sind unten aufgeführt.

Konvertieren von Bezeichner in Zeichenfolge

Mit dieser Funktion kann auch ein Bezeichner in `string` konvertiert werden, indem der Bezeichner im Argument `Parameter` für jedes Datenobjekt mit dem Gültigkeitsbereich global, lokal in Modul oder lokal in Routine angegeben wird:

```
VAR num chales :=5;
...
proc1;

PROC proc1 ()
  VAR string name;
  ...
  name:=ArgName(chales);
  TPWrite "Global data object "+name+" has value "\Num:=chales;
ENDPROC
```

Der Variablen `name` wird die Zeichenfolge "`chales`" zugewiesen und am FlexPendant wird folgende Zeichenfolge ausgegeben: "Global data object chales has value 5".

Routineaufruf in mehreren Schritten

Beachten Sie, dass die Funktion den **originalen** Datenobjektnamen zurückgibt:

```
VAR num chales :=5;
...
proc1 chales;
...
PROC proc1 (num parameter1)
  ...
  proc2 parameter1;
  ...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
ENDPROC

PROC proc2 (num par1)
  VAR string name;
  ...
  name:=ArgName(par1);
  TPWrite "Original data object name "+name+" with value"
       \Num:=par1;
ENDPROC
```

Der Variablen `name` wird die Zeichenfolge "`chales`" zugewiesen und am FlexPendant wird folgende Zeichenfolge ausgegeben: "Original data object name chales with value 5".

Abarbeitung in Fehlerbehandlung unterdrücken

```
PROC main()
  VAR string mystring:="DUMMY";
  proc1 mystring;
  proc1 "This is a test";
  ...
ENDPROC

PROC proc1 (string par1)
  VAR string name;
  VAR errnum myerrnum;

  name := ArgName(par1 \ErrorNumber:=myerrnum);
  IF myerrnum=ERR_ARGNAME THEN
    TPWrite "The argument par1 is an expression value";
    TPWrite "The name of the argument can not be evaluated";
  ELSE
    TPWrite "The name on the argument is "+name;
  ENDIF
ENDPROC
```

Die Variable `name`wird dem Zeichenfolgenwert "`mystring`" zugewiesen, wenn der erste Aufruf für `proc1` erfolgt. Wenn der zweite Aufruf für `proc1` erfolgt, wird 'name' eine leere Zeichenfolge zugewiesen, und auf dem FlexPendant wird folgende Zeichenfolge geschrieben: "`The argument par1 is an expression value`" und "`The name of the argument can not be evaluated`".

Syntax

```
ArgName '('
  [ Parameter ':=' ] < reference (REF) of anytype>
  ['\' ErrorNumber ':=' <var or pers (INOUT) of errnum>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.8 ArgName - Abrufen des Argumentnamens

RobotWare - OS

Fortsetzung

Information über	Siehe
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Advanced RAPID*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.9 ASin - Berechnen den Arcussinus

Verwendung

`ASin` (*Arc Sine*) wird zur Berechnung des Arcussinuswertes für die Datentypen `num` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ASin`

Beispiel 1:

```
VAR num angle;
VAR num value;
...
...
angle := ASin(value);
```

`angle` will get the arc sine value of `value`

Rückgabewert

Datentyp: `num`

Der Arcussinuswert in Grad, Bereich [-90, 90].

Argumente

```
ASin (Value)
```

`Value`

Datentyp: `num`

Der Argumentwert muss im Bereich [-1, 1] liegen.

Einschränkungen

Die Abarbeitung der Funktion `ASin(x)` ergibt einen Fehler, wenn x außerhalb des Bereichs [1, -1] liegt.

Syntax

```
ASin '('
  [Value ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.10 ASinDnum - Berechnen den Arcussinus

Verwendung

`ASinDnum` (*Arc Sine dnum*) wird zur Berechnung des Arcussinuswertes für die Datentypen `dnum` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ASinDnum`

Beispiel 1:

```
VAR dnum angle;
VAR dnum value;
...
...
angle := ASinDnum(value);
```

`angle` will get the arc sine value of `value`

Rückgabewert

Datentyp: `dnum`

Der Arcussinuswert in Grad, Bereich [-90, 90].

Argumente

```
ASinDnum (Value)
```

`Value`

Datentyp: `dnum`

Der Argumentwert muss im Bereich [-1, 1] liegen.

Einschränkungen

Die Abarbeitung der Funktion `ASinDnum(x)` ergibt einen Fehler, wenn x außerhalb des Bereichs [1, -1] liegt.

Syntax

```
ASinDnum '('
  [Value ':='] <expression (IN) of dnum> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.11 ATan - Berechnen des Arcustangens

Verwendung

`ATan` (*Arc Tangent*) wird zur Berechnung des Arcustangenswertes für die Datentypen `num` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ATan`.

Beispiel 1:

```
VAR num angle;
VAR num value;
...
...
angle := ATan(value);
```

`angle` ruft den Arcustangens-Wert von `value` ab.

Rückgabewert

Datentyp: `num`

Der Arcustangenswert in Grad, Bereich [-90, 90].

Argumente

```
ATan (Value)
```

`Value`

Datentyp: `num`

Der Argumentwert.

Syntax

```
ATan '('
  [Value ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Arcustangens mit einem Rückgabewert im Bereich [-180, 180]	*ATan2 - Berechnen des Arcustangens2 auf Seite 1187*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.12 ATanDnum - Berechnen des Arcustangens

Verwendung

ATanDnum (*Arc Tangent dnum*) wird zur Berechnung des Arcustangenswertes für die Datentypen dnum verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion ATanDnum.

Beispiel 1:

```
VAR dnum angle;
VAR dnum value;
...
...
angle := ATanDnum(value);
```

angle ruft den Arcustangens-Wert von value ab.

Rückgabewert

Datentyp: dnum

Der Arcustangenswert in Grad, Bereich [-90, 90].

Argumente

```
ATanDnum (Value)
```

Value

Datentyp: dnum

Der Argumentwert.

Syntax

```
ATanDnum '('
  [Value ':='] <expression (IN) of dnum> ')'
```

Eine Funktion mit einem Rückgabewert des Typs dnum.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Arcustangens mit einem Rückgabewert im Bereich [-180, 180]	*ATan2 - Berechnen des Arcustangens2 auf Seite 1187*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.13 ATan2 - Berechnen des Arcustangens2

Verwendung

ATan2 (*Arc Tangent2*) wird zur Berechnung des Arcustangens2wertes für die Datentypen num verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion ATan2.

Beispiel 1:

```
VAR num angle;
VAR num x_value;
VAR num y_value;
...
...
angle := ATan2(y_value, x_value);
```

angle ruft den Arcustangens-Wert von y_value/x_value ab.

Rückgabewert

Datentyp: num

Der Arcustangenswert in Grad, Bereich [-180, 180]. Der Wert ist gleich ATan(y/x), jedoch im Bereich [-180, 180], da die Funktion die Vorzeichen beider Argumente verwendet, um den Quadranten für den Rückgabewert zu bestimmen.

Argumente

```
ATan2 (Y X)
```

Y

Datentyp: num

Der Zähler des Argumentwerts.

X

Datentyp: num

Der Nenner des Argumentwerts.

Syntax

```
ATan2 '('
  [Y ':='] <expression (IN) of num> ','
  [X ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Arcustangens mit nur einem Argument	*ATan - Berechnen des Arcustangens auf Seite 1185*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.14 ATan2Dnum - Berechnen des Arcustangens2

Verwendung

`ATan2Dnum` (*Arc Tangent2 dnum*) wird zur Berechnung des Arcustangens2wertes für die Datentypen `dnum` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ATan2Dnum`.

Beispiel 1:

```
VAR dnum angle;
VAR dnum x_value;
VAR dnum y_value;
...
...
angle := ATan2Dnum(y_value, x_value);
```

`angle` ruft den Arcustangens-Wert von `y_value/x_value` ab.

Rückgabewert

Datentyp: `dnum`

Der Arcustangenswert in Grad, Bereich [-180, 180]. Der Wert ist gleich ATanDnum(y/x), jedoch im Bereich [-180, 180], da die Funktion die Vorzeichen beider Argumente verwendet, um den Quadranten für den Rückgabewert zu bestimmen.

Argumente

```
ATan2Dnum (Y X)
```

Y

Datentyp: `dnum`

Der Zähler des Argumentwerts.

X

Datentyp: `dnum`

Der Nenner des Argumentwerts.

Syntax

```
ATan2Dnum '('
  [Y ':='] <expression (IN) of dnum> ','
  [X ':='] <expression (IN) of dnum> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Arcustangens mit nur einem Argument	*ATan - Berechnen des Arcustangens auf Seite 1185*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.15 BitAnd - Logische bitweise AND-Operation mit Byte-Daten

Verwendung

`BitAnd` wird verwendet, um eine logische bitweise `AND`-Operation mit byte-Datentypen abzuarbeiten.

Grundlegende Beispiele

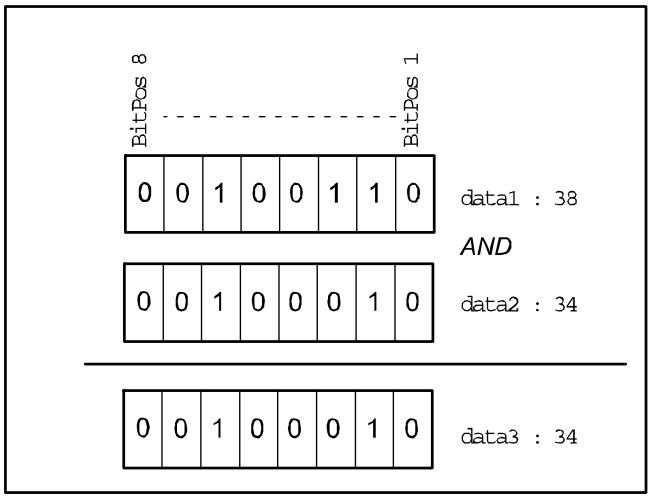
Das folgende Beispiel veranschaulicht die Funktion `BitAnd`.

Beispiel 1:

```
VAR byte data1 := 38;
VAR byte data2 := 34;
VAR byte data3;

data3 := BitAnd(data1, data2);
```

Die logische bitweise AND-Operation (siehe nachfolgende Abbildung) wird auf `data1` und `data2` ausgeführt. Das Ergebnis wird an `data3` ausgegeben (ganzzahlige Darstellung).



xx0500002454

Rückgabewert

Datentyp: `byte`

Das Ergebnis der logischen bitweisen `AND`-Operation in ganzzahliger Darstellung.

Argumente

```
BitAnd (BitData1 BitData2)
```

`BitData1`

Datentyp: `byte`

Die Bitdaten 1 in ganzzahliger Darstellung.

`BitData2`

Datentyp: `byte`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.15 BitAnd - Logische bitweise AND-Operation mit Byte-Daten

RobotWare - OS

Fortsetzung

Die Bitdaten 2 in ganzzahliger Darstellung.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255.

Syntax

```
BitAnd '('
  [BitData1 ':='] <expression (IN) of byte> ','
  [BitData2 ':='] <expression (IN) of byte> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `byte`.

Weitere Informationen

Information über	Siehe
Logische bitweise OR-Operation mit byte-Daten	*BitOr - Logische bitweise OR-Operation mit Byte-Daten auf Seite 1206*
Logische bitweise XOR-Operation mit byte-Daten	*BitXOr - Logische bitweise XOR-Operation mit Byte-Daten auf Seite 1214*
Logische bitweise NEGATION-Operation mit byte-Daten	*BitNeg - Logische bitweise NEGATION-Operation mit Byte-Daten auf Seite 1202*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.16 BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten

Verwendung

`BitAndDnum` wird verwendet, um eine logische bitweise `AND`-Operation mit `dnum`-Datentypen abzuarbeiten.

Grundlegende Beispiele

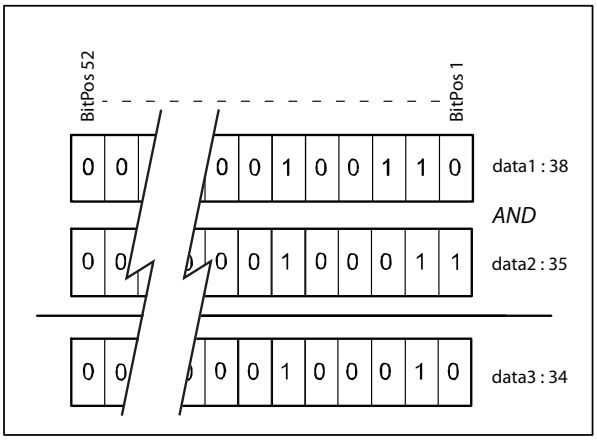
Das folgende Beispiel veranschaulicht die Funktion `BitAndDnum`.

Beispiel 1:

```
VAR dnum data1 := 38;
VAR dnum data2 := 35;
VAR dnum data3;

data3 := BitAndDnum(data1, data2);
```

Die logische bitweise AND-Operation (siehe Abbildung unten) wird mit `data1` und `data2` abgearbeitet. Das Ergebnis wird an `data3` (ganzzahlige Darstellung) zurückgegeben.



xx1200000007

Rückgabewert

Datentyp: `dnum`

Das Ergebnis der logischen bitweisen `AND`-Operation in ganzzahliger Darstellung.

Argumente

```
BitAndDnum (Value1 Value2)
```

`Value1`

Datentyp: `dnum`

Der erste Bitdatenwert in ganzzahliger Darstellung.

`Value2`

Datentyp: `dnum`

Der zweite Bitdatenwert in ganzzahliger Darstellung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.16 BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten

RobotWare - OS

Fortsetzung

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495.

Syntax

```
BitAndDnum '('
  [Value1 ':='] <expression (IN) of dnum> ','
  [Value2 ':='] <expression (IN) of dnum> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Logische bitweise AND-Operation mit **byte**-Daten	*BitAnd - Logische bitweise AND-Operation mit Byte-Daten auf Seite 1189*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Logische bitweise OR-Operation mit `dnum`-Daten	*BitOrDnum – Logische bitweise OR-Operation mit dnum-Daten auf Seite 1208*
Logische bitweise XOR-Operation mit `dnum`-Daten	*BitXOrDnum – Logische bitweise XOR-Operation mit dnum-Daten auf Seite 1216*
Logische bitweise NEGATION-Operation mit `dnum`-Daten	*BitNegDnum – Logische bitweise NEGATION-Operation mit dnum-Daten auf Seite 1204*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.17 BitCheck - Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist

Verwendung

`BitCheck` wird verwendet, um zu prüfen, ob ein angegebenes Bit in definierten `byte`-Daten auf 1 gesetzt ist.

Grundlegende Beispiele

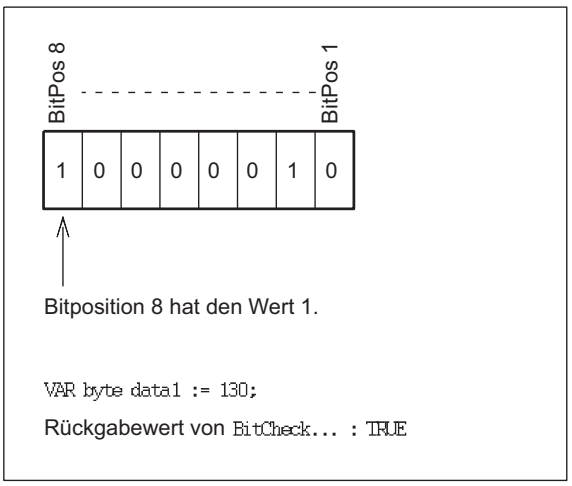
Das folgende Beispiel veranschaulicht die Funktion `BitCheck`.

Beispiel 1:

```
CONST num parity_bit := 8;
VAR byte data1 := 130;

IF BitCheck(data1, parity_bit) = TRUE THEN
  ...
ELSE
  ...
ENDIF
```

Die Bitnummer `8` (`parity_bit`) in der Variablen `data1` wird überprüft, wenn z. B. das angegebene Bit auf 1 in der Variable `data1` gesetzt wird, dann kehrt diese Funktion auf `TRUE` zurück. Die Bitüberprüfung von Datentyp `byte` wird in der folgenden Abbildung gezeigt.



xx0500002442

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn das angegebene Bit auf 1 gesetzt ist; `FALSE`, wenn das angegebene Bit auf 0 gesetzt ist.

Argumente

```
BitCheck (BitData BitPos)
```

`BitData`

Datentyp: `byte`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die zu prüfenden Bitdaten in ganzzahliger Darstellung.

`BitPos`

Bit Position

Datentyp: `num`

Die zu prüfende Bitposition (1-8) in `BitData`.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255 dezimal.

Für die Bitposition ist ein Wert von 1-8 gültig.

Syntax

```
BitCheck '('
  [BitData ':='] <expression (IN) of byte> ','
  [BitPos ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Setzen eines angegebenen Bits in Byte-Daten	*BitSet – Setzen eines angegebenen Bits in Byte- oder dnum-Daten auf Seite 47*
Löschen eines angegebenen Bits in einem Byte	*BitClear – Löschen eines angegebenen Bits in Byte- oder dnum-Daten auf Seite 44*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.18 BitCheckDnum – Prüfen, ob ein angegebenes Bit in den dnum-Daten gesetzt ist

Verwendung

`BitCheckDnum` wird verwendet, um zu prüfen, ob ein angegebenes Bit in definierten `dnum`-Daten auf 1 gesetzt ist.

Grundlegende Beispiele

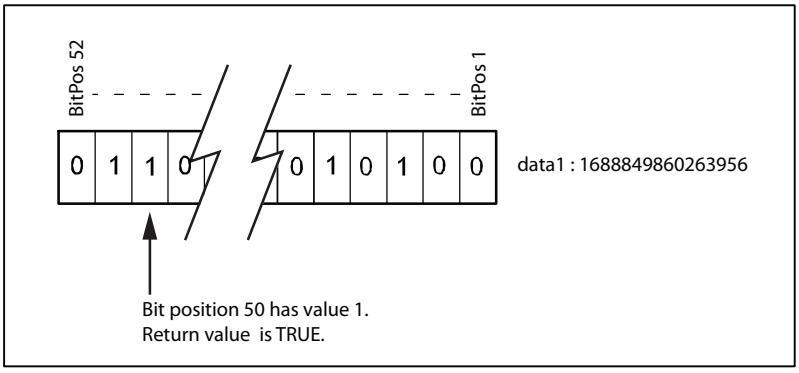
Das folgende Beispiel veranschaulicht die Funktion `BitCheckDnum`.

Beispiel 1:

```
CONST num check_bit := 50;
VAR dnum data1 := 1688849860263956;

IF BitCheckDnum(data1, check_bit) = TRUE THEN
  ...
ELSE
  ...
ENDIF
```

Die Bitnummer `50` (`check_bit`) in der Variablen `data1` wird überprüft, wenn z. B. das angegebene Bit in der Variablen `data1` 1 ist, dann kehrt diese Funktion auf `TRUE` zurück. Die Bitüberprüfung von Datentyp `dnum` wird in der Abbildung unten gezeigt.



xx1200000016

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn das angegebene Bit auf 1 gesetzt ist; `FALSE`, wenn das angegebene Bit auf 0 gesetzt ist.

Argumente

```
BitCheckDnum (Value BitPos)
```

`Value`

Datentyp: `dnum`

Die zu prüfenden Bitdaten in ganzzahliger Darstellung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.18 BitCheckDnum – Prüfen, ob ein angegebenes Bit in den dnum-Daten gesetzt ist

RobotWare - OS

Fortsetzung

`BitPos`

Bit Position

Datentyp: `num`

Die zu prüfende Bitposition (1-52) in `Value`.

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495 dezimal.

Für die Bitposition ist ein Wert von 1-52 gültig.

Syntax

```
BitCheckDnum '('
  [Value ':='] <expression (IN) of dnum> ','
  [BitPos ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Prüfen, ob ein angegebenes Bit in den **byte**-Daten gesetzt ist	*BitCheck - Prüfen, ob ein angegebenes Bit in einem Byte gesetzt ist auf Seite 1193*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Setzen eines angegebenen Bits in `byte`- oder `dnum`-Daten	*BitSet – Setzen eines angegebenen Bits in Byte- oder dnum-Daten auf Seite 47*
Löschen eines angegebenen Bits in `byte`- oder `dnum`-Daten	*BitClear – Löschen eines angegebenen Bits in Byte- oder dnum-Daten auf Seite 44*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.19 BitLSh - Logische bitweise LEFT SHIFT-Operation mit Byte-Daten

Verwendung

`BitLSh` (***Bit Left Shift***) wird verwendet, um eine logische bitweise LEFT SHIFT-Operation mit `byte`-Datentypen abzuarbeiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitLSh`.

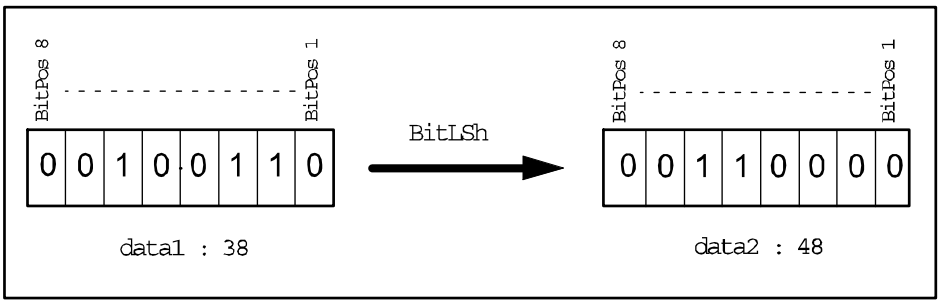
Beispiel 1:

```
VAR num left_shift := 3;
VAR byte data1 := 38;
VAR byte data2;

data2 := BitLSh(data1, left_shift);
```

Die logische bitweise LEFT SHIFT-Operation wird an `data1` mit `3` Linksverschiebungsschritten (`left_shift`) ausgeführt. Das Ergebnis wird an `data2` (ganzzahlige Darstellung) zurückgegeben.

Die folgende Abbildung zeigt eine logische bitweise LEFT SHIFT-Operation.



xx0500002457

Rückgabewert

Datentyp: `byte`

Das Ergebnis der logischen bitweisen LEFT SHIFT-Operation in ganzzahliger Darstellung.

Die Bitstellen rechts werden mit 0-Bits aufgefüllt.

Argumente

```
BitLSh (BitData ShiftSteps)
```

`BitData`

Datentyp: `byte`

Die zu verschiebenden Bitdaten in ganzzahliger Darstellung.

`ShiftSteps`

Datentyp: `num`

Anzahl der logischen Verschiebungen (0 - 8), die ausgeführt werden sollen.

Die Verwendung von 0 gibt den Wert im Argument `BitData` zurück.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.19 BitLSh - Logische bitweise LEFT SHIFT-Operation mit Byte-Daten

RobotWare - OS

Fortsetzung

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255.

Das `ShiftSteps` ist gemäß einem Byte von 1-8 gültig.

Syntax

```
BitLSh '('
  [BitData ':='] <expression (IN) of byte> ','
  [ShiftSteps ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `byte`.

Weitere Informationen

Information über	Siehe
Logische bitweise RIGHT SHIFT-Operation mit Byte-Daten	*BitRSh - Logische bitweise RIGHT SHIFT-Operation mit Byte-Daten auf Seite 1210*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.20 BitLShDnum – Logische bitweise LEFT SHIFT-Operation mit dnum-Daten

Verwendung

`BitLShDnum` (*Bit Left Shift dnum*) wird verwendet, um eine logische bitweise LEFT SHIFT-Operation mit `dnum`-Datentypen abzuarbeiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitLShDnum`.

Siehe auch *Weitere Beispiele auf Seite 1200*.

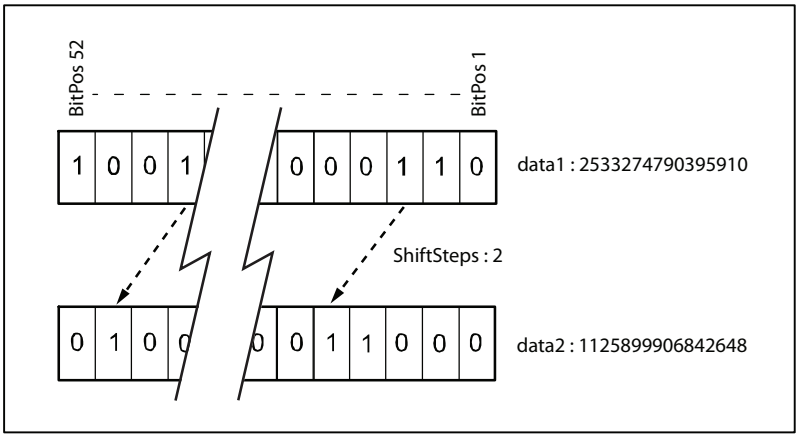
Beispiel 1:

```
VAR num left_shift := 2;
VAR dnum data1 := 2533274790395910;
VAR dnum data2;

data2 := BitLShDnum(data1, left_shift);
```

Die logische bitweise LEFT SHIFT-Operation wird an `data1` mit `2` Linksverschiebungsschritten (`left_shift`) ausgeführt. Das Ergebnis wird an `data2` (ganzzahlige Darstellung) zurückgegeben.

Die folgende Abbildung zeigt eine logische bitweise LEFT SHIFT-Operation.



xx1200000008

Rückgabewert

Datentyp: `dnum`

Das Ergebnis der logischen bitweisen LEFT SHIFT-Operation in ganzzahliger Darstellung.

Die Bitstellen rechts werden mit 0-Bits aufgefüllt.

Argumente

```
BitLShDnum (Value ShiftSteps [\Size])
```

`Value`

Datentyp: `dnum`

Die zu verschiebenden Bitdaten in ganzzahliger Darstellung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.20 BitLShDnum – Logische bitweise LEFT SHIFT-Operation mit dnum-Daten

RobotWare - OS

Fortsetzung

`ShiftSteps`

Datentyp: `num`

Anzahl der logischen Verschiebungen (0 - 52), die ausgeführt werden sollen.

Die Verwendung von 0 gibt den Wert im Argument `Value` zurück.

`Size`

Datentyp: `num`

Die Größe (Anzahl der Bits), die bei der Durchführung der logischen bitweisen LEFT SHIFT-Operation mit dem Argument `Value` berücksichtigt werden sollte. Für die Größe ist ein Wert von 1-52 gültig.

Weitere Beispiele

Weitere Beispiele für die Funktion `BitLshDnum` sind unten aufgeführt.

Beispiel 1:

```
VAR dnum result;
VAR dnum data1:=221;
! Only consider the 8 lowest bits
result := BitLshDnum(data1, 4 \Size:=8);
TPWrite "" \Dnum:=result;
! Consider all 52 bits in the dnum datatype
result := BitLshDnum(data1, 4);
TPWrite "" \Dnum:=result;
```

Die logische bitweise LEFT SHIFT-Operation wird mit `data1` ausgeführt und das Ergebnis an `result` (ganzzahlige Darstellung) zurückgegeben. Der erste am FlexPendant angezeigte Wert ist 208. Der zweite Wert am FlexPendant beträgt 3536.

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495.

Das `ShiftSteps`-Argument ist von 1-52 gültig, da ein `dnum` 52 Bits entspricht.

Syntax

```
BitLShDnum '('
  [Value ':='] <expression (IN) of dnum> ','
  [ShiftSteps ':='] <expression (IN) of num>
  ['\' Size ':=' < expression (IN) of num>]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Logische bitweise LEFT SHIFT-Operation mit **byte**-Daten	*BitLSh - Logische bitweise LEFT SHIFT-Operation mit Byte-Daten auf Seite 1197*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Logische bitweise RIGHT SHIFT-Operation mit dnum-Daten	*BitRShDnum – Logische bitweise RIGHT SHIFT-Operation mit dnum-Daten auf Seite 1212*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.20 BitLShDnum – Logische bitweise LEFT SHIFT-Operation mit dnum-Daten

RobotWare - OS

Fortsetzung

Information über	Siehe
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.21 BitNeg - Logische bitweise NEGATION-Operation mit Byte-Daten

Verwendung

`BitNeg` (*Bit Negation*) wird verwendet, um eine logische bitweise `NEGATION`-Operation (Einserkomplement) mit dem Datentyp `byte` auszuführen.

Grundlegende Beispiele

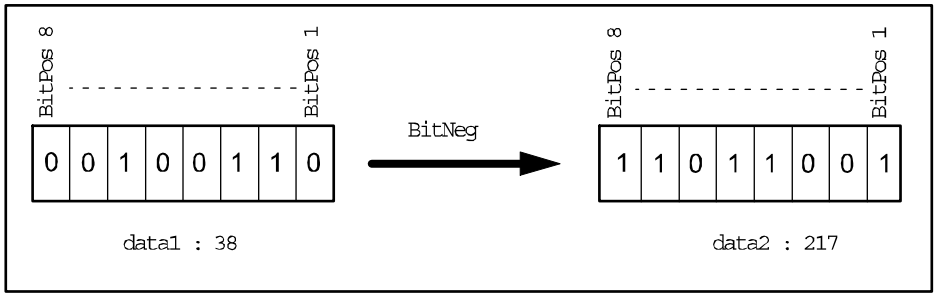
Das folgende Beispiel veranschaulicht die Funktion `BitNeg`.

Beispiel 1:

```
VAR byte data1 := 38;
VAR byte data2;

data2 := BitNeg(data1);
```

Die logische bitweise `NEGATION`-Operation (siehe Abbildung unten) wird mit `data1` abgearbeitet, das Ergebnis wird an `data2` (ganzzahlige Darstellung) zurückgegeben.



xx0500002456

Rückgabewert

Datentyp: `byte`

Das Ergebnis der logischen bitweisen `NEGATION`-Operation in ganzzahliger Darstellung.

Argumente

```
BitNeg (BitData)
```

`BitData`

Datentyp: `byte`

Die Bytedaten in ganzzahliger Darstellung.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255.

Syntax

```
BitNeg '('
  [BitData ':='] <expression (IN) of byte>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `byte`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Logische bitweise AND-Operation mit Byte-Daten	*BitAnd - Logische bitweise AND-Operation mit Byte-Daten auf Seite 1189*
Logische bitweise OR-Operation mit byte-Daten	*BitOr - Logische bitweise OR-Operation mit Byte-Daten auf Seite 1206*
Logische bitweise XOR-Operation mit byte-Daten	*BitXOr - Logische bitweise XOR-Operation mit Byte-Daten auf Seite 1214*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.22 BitNegDnum – Logische bitweise NEGATION-Operation mit dnum-Daten

Verwendung

`BitNegDnum` (*Bit Negation dnum*) wird verwendet, um eine logische bitweise `NEGATION`-Operation (Einserkomplement) mit dem Datentyp `dnum` auszuführen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitNegDnum`.

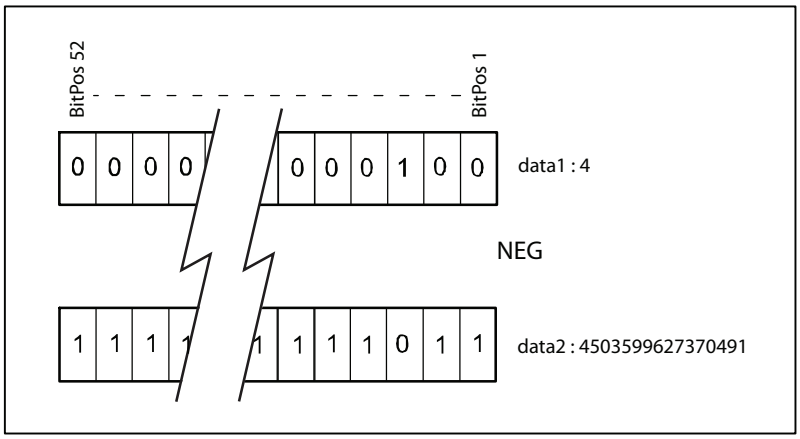
Siehe auch *Weitere Beispiele auf Seite 1205*.

Beispiel 1:

```
VAR dnum data1 := 4;
VAR dnum data2;

data2 := BitNegDnum(data1);
```

Die logische bitweise `NEGATION`-Operation (siehe Abbildung unten) wird mit `data1` abgearbeitet, das Ergebnis wird an `data2` (ganzzahlige Darstellung) zurückgegeben.



xx1200000012

Rückgabewert

Datentyp: `dnum`

Das Ergebnis der logischen bitweisen `NEGATION`-Operation in ganzzahliger Darstellung.

Argumente

```
BitNegDnum (Value [\Size])
```

`Value`

Datentyp: `dnum`

Die dnum-Daten in ganzzahliger Darstellung.

`Size`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Größe (Anzahl der Bits), die bei der Durchführung der logischen bitweisen NEGATION-Operation mit dem Argument `Value` berücksichtigt werden sollte. Für die Größe ist ein Wert von 1-52 gültig.

Weitere Beispiele

Weitere Beispiele für die Funktion `BitNegDnum` sind unten aufgeführt.

Beispiel 1:

```
VAR dnum result;
VAR dnum data1:=38;
! Only consider the 16 lowest bits
result := BitNegDnum(data1 \Size:=16);
TPWrite "" \Dnum:=result;
! Consider all 52 bits in the dnum datatype
result := BitNegDnum(data1);
TPWrite "" \Dnum:=result;
```

Die logische bitweise NEGATION-Operation wird mit `data1` ausgeführt und das Ergebnis an `result` (ganzzahlige Darstellung) zurückgegeben. Der erste am FlexPendant angezeigte Wert ist 65497. Der zweite Wert am FlexPendant beträgt 4503599627370457.

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495.

Syntax

```
BitNegDnum '('
  [Value ':='] <expression (IN) of dnum>
  ['\'Size ':=' < expression (IN) of num>]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Logische bitweise NEGATION-Operation mit **byte**-Daten	*BitNeg - Logische bitweise NEGATION-Operation mit Byte-Daten auf Seite 1202*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Logische bitweise AND-Operation mit dnum-Daten	*BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten auf Seite 1191*
Logische bitweise OR-Operation mit dnum-Daten	*BitOrDnum – Logische bitweise OR-Operation mit dnum-Daten auf Seite 1208*
Logische bitweise XOR-Operation mit dnum-Daten	*BitXOrDnum – Logische bitweise XOR-Operation mit dnum-Daten auf Seite 1216*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.23 BitOr - Logische bitweise OR-Operation mit Byte-Daten

Verwendung

`BitOr` (*Bit inclusive Or*) wird verwendet, um eine logische bitweise `OR`-Operation mit `byte`-Datentypen abzuarbeiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitOr`.

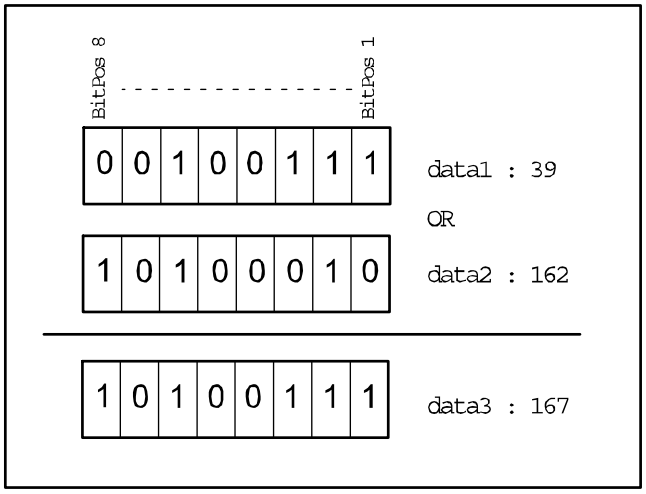
Beispiel 1:

```
VAR byte data1 := 39;
VAR byte data2 := 162;
VAR byte data3;

data3 := BitOr(data1, data2);
```

Die logische bitweise OR-Operation wird mit `data1` und `data2` abgearbeitet und das Ergebnis wird an `data3` (ganzzahlige Darstellung) zurückgegeben.

Die folgende Abbildung zeigt eine logische bitweise OR-Operation.



xx0500002458

Rückgabewert

Datentyp: `byte`

Das Ergebnis der logischen bitweisen OR-Operation in ganzzahliger Darstellung.

Argumente

```
BitOr (BitData1 BitData2)
```

`BitData1`

Datentyp: `byte`

Die Bitdaten 1 in ganzzahliger Darstellung.

`BitData2`

Datentyp: `byte`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Bitdaten 2 in ganzzahliger Darstellung.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255.

Syntax

```
BitOr '('
  [BitData1 ':='] <expression (IN) of byte> ','
  [BitData2 ':='] <expression (IN) of byte>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `byte`.

Weitere Informationen

Information über	Siehe
Logische bitweise AND-Operation mit Byte-Daten	*BitAnd - Logische bitweise AND-Operation mit Byte-Daten auf Seite 1189*
Logische bitweise XOR-Operation mit byte-Daten	*BitXOr - Logische bitweise XOR-Operation mit Byte-Daten auf Seite 1214*
Logische bitweise NEGATION-Operation mit byte-Daten	*BitNeg - Logische bitweise NEGATION-Operation mit Byte-Daten auf Seite 1202*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.24 BitOrDnum – Logische bitweise OR-Operation mit dnum-Daten

Verwendung

`BitOrDnum` (*Bit inclusive Or dnum*) wird verwendet, um eine logische bitweise `OR`-Operation mit `dnum`-Datentypen abzuarbeiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitOrDnum`.

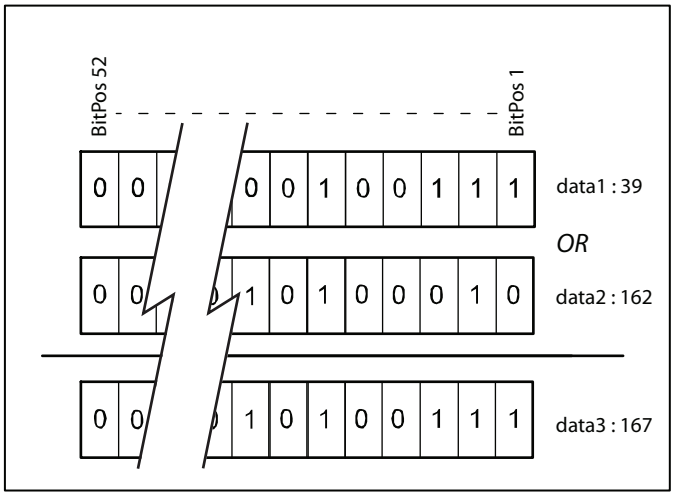
Beispiel 1:

```
VAR dnum data1 := 39;
VAR dnum data2 := 162;
VAR dnum data3;

data3 := BitOrDnum(data1, data2);
```

Die logische bitweise OR-Operation wird mit `data1` und `data2` abgearbeitet und das Ergebnis wird an `data3` (ganzzahlige Darstellung) zurückgegeben.

Die folgende Abbildung zeigt eine logische bitweise OR-Operation.



xx1200000011

Rückgabewert

Datentyp: `dnum`

Das Ergebnis der logischen bitweisen OR-Operation in ganzzahliger Darstellung.

Argumente

```
BitOrDnum (Value1 Value2)
```

`Value1`

Datentyp: `dnum`

Der erste Bitdatenwert in ganzzahliger Darstellung.

`Value2`

Datentyp: `dnum`

Der zweite Bitdatenwert in ganzzahliger Darstellung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495.

Syntax

```
BitOrDnum '('
  [Value1 ':='] <expression (IN) of dnum> ','
  [Value2 ':='] <expression (IN) of dnum>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Logische bitweise OR-Operation mit **byte**-Daten	*BitOr - Logische bitweise OR-Operation mit Byte-Daten auf Seite 1206*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Logische bitweise AND-Operation mit `dnum`-Daten	*BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten auf Seite 1191*
Logische bitweise XOR-Operation mit `dnum`-Daten	*BitXOrDnum – Logische bitweise XOR-Operation mit dnum-Daten auf Seite 1216*
Logische bitweise NEGATION-Operation mit `dnum`-Daten	*BitNegDnum – Logische bitweise NEGATION-Operation mit dnum-Daten auf Seite 1204*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.25 BitRSh - Logische bitweise RIGHT SHIFT-Operation mit Byte-Daten

Verwendung

`BitRSh` (*Bit Right Shift*) wird verwendet, um eine logische bitweise RIGHT SHIFT-Operation mit `byte`-Datentypen auszuführen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitRSh`.

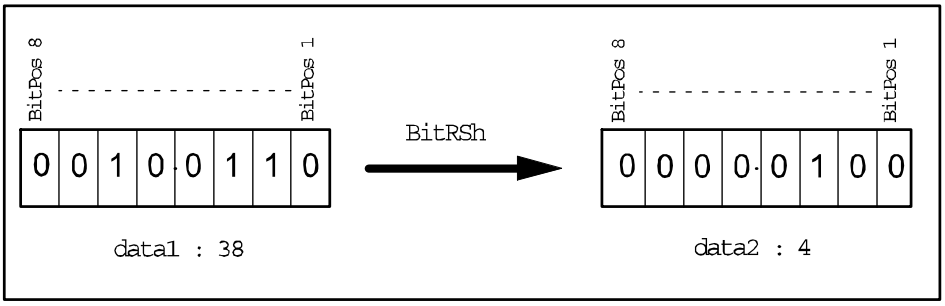
Beispiel 1:

```
VAR num right_shift := 3;
VAR byte data1 := 38;
VAR byte data2;

data2 := BitRSh(data1, right_shift);
```

Die logische bitweise RIGHT SHIFT-Operation wird an `data1` mit `3` (`right_shift`) Rechtsverschiebungsschritten ausgeführt. Das Ergebnis wird an `data2` (ganzzahlige Darstellung) zurückgegeben

Die folgende Abbildung zeigt eine logische bitweise RIGHT SHIFT-Operation.



xx0500002455

Rückgabewert

Datentyp: `byte`

Das Ergebnis der logischen bitweisen RIGHT SHIFT-Operation in ganzzahliger Darstellung.

Die Bitstellen links werden mit 0-Bits aufgefüllt.

Argumente

```
BitRSh (BitData ShiftSteps)
```

`BitData`

Datentyp: `byte`

Die zu verschiebenden Bitdaten in ganzzahliger Darstellung.

`ShiftSteps`

Datentyp: `num`

Anzahl der logischen Verschiebungen (0 - 8), die ausgeführt werden sollen.

Die Verwendung von 0 gibt den Wert im Argument `BitData` zurück.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255.

Das `ShiftSteps` ist gemäß einem Byte von 1-8 gültig.

Syntax

```
BitRSh '('
  [BitData ':='] <expression (IN) of byte> ','
  [ShiftSteps ':='] <expression (IN) of num>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs byte.

Weitere Informationen

Information über	Siehe
Logische bitweise LEFT SHIFT-Operation mit Byte-Daten	*BitLSh - Logische bitweise LEFT SHIFT-Operation mit Byte-Daten auf Seite 1197*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.26 BitRShDnum – Logische bitweise RIGHT SHIFT-Operation mit dnum-Daten

Verwendung

`BitRShDnum` (*Bit Right Shift dnum*) wird verwendet, um eine logische bitweise RIGHT SHIFT-Operation mit `dnum`-Datentypen auszuführen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitRShDnum`.

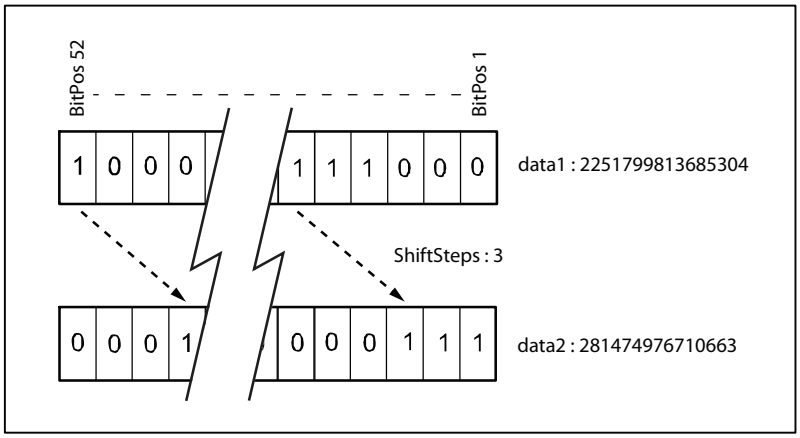
Beispiel 1:

```
VAR num right_shift := 3;
VAR dnum data1 := 2251799813685304;
VAR dnum data2;

data2 := BitRShDnum(data1, right_shift);
```

Die logische bitweise RIGHT SHIFT-Operation wird an `data1` mit `3` (`right_shift`) Rechtsverschiebungsschritten ausgeführt. Das Ergebnis wird an `data2` (ganzzahlige Darstellung) zurückgegeben

Die folgende Abbildung zeigt eine logische bitweise RIGHT SHIFT-Operation.



xx1200000009

Rückgabewert

Datentyp: `dnum`

Das Ergebnis der logischen bitweisen RIGHT SHIFT-Operation in ganzzahliger Darstellung.

Die Bitstellen links werden mit 0-Bits aufgefüllt.

Argumente

```
BitRShDnum (Value ShiftSteps)
```

`Value`

Datentyp: `dnum`

Die zu verschiebenden Bitdaten in ganzzahliger Darstellung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`ShiftSteps`

Datentyp: `num`

Anzahl der logischen Verschiebungen (0 - 52), die ausgeführt werden sollen.

Die Verwendung von 0 gibt den Wert im Argument `Value` zurück.

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495.

Das `ShiftSteps`-Argument ist von 1-52 gültig, da ein dnum 52 Bits entspricht.

Syntax

```
BitRShDnum '('
  [Value ':='] <expression (IN) of dnum> ','
  [ShiftSteps ':='] <expression (IN) of num>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Logische bitweise RIGHT SHIFT-Operation mit **byte**-Daten	*BitRSh - Logische bitweise RIGHT SHIFT-Operation mit Byte-Daten auf Seite 1210*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Logische bitweise LEFT SHIFT-Operation mit dnum-Daten	*BitLShDnum – Logische bitweise LEFT SHIFT-Operation mit dnum-Daten auf Seite 1199*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.27 BitXOr - Logische bitweise XOR-Operation mit Byte-Daten

Verwendung

`BitXOr` (*Bit eXclusive Or*) wird verwendet, um eine logische bitweise XOR-Operation mit `byte`-Datentypen abzuarbeiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitXOr`.

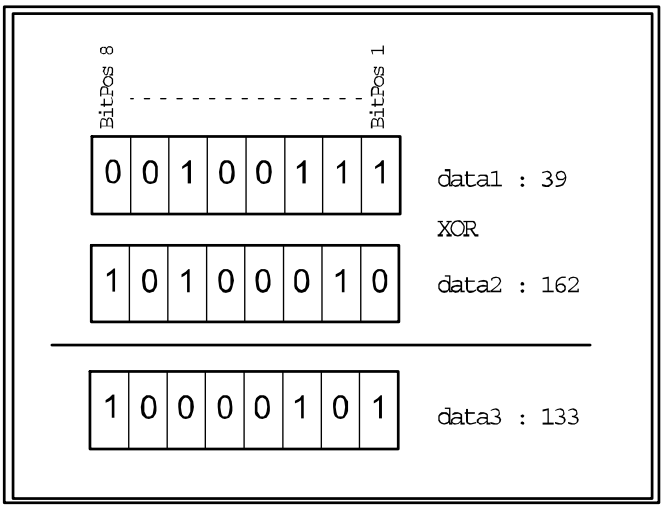
Beispiel 1:

```
VAR byte data1 := 39;
VAR byte data2 := 162;
VAR byte data3;

data3 := BitXOr(data1, data2);
```

Die logische bitweise XOR-Operation wird mit `data1` und `data2` abgearbeitet und das Ergebnis wird an `data3` (ganzzahlige Darstellung) zurückgegeben.

Die folgende Abbildung zeigt eine logische bitweise XOR-Operation.



xx0500002459

Rückgabewert

Datentyp: `byte`

Das Ergebnis der logischen bitweisen XOR-Operation in ganzzahliger Darstellung.

Argumente

```
BitXOr (BitData1 BitData2)
```

`BitData1`

Datentyp: `byte`

Die Bitdaten 1 in ganzzahliger Darstellung.

`BitData2`

Datentyp: `byte`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Bitdaten 2 in ganzzahliger Darstellung.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255.

Syntax

```
BitXOr '('
  [BitData1 ':='] <expression (IN) of byte> ','
  [BitData2 ':='] <expression (IN) of byte>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `byte`.

Weitere Informationen

Information über	Siehe
Logische bitweise AND-Operation mit Byte-Daten	*BitAnd - Logische bitweise AND-Operation mit Byte-Daten auf Seite 1189*
Logische bitweise OR-Operation mit byte-Daten	*BitOr - Logische bitweise OR-Operation mit Byte-Daten auf Seite 1206*
Logische bitweise NEGATION-Operation mit byte-Daten	*BitNeg - Logische bitweise NEGATION-Operation mit Byte-Daten auf Seite 1202*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.28 BitXOrDnum – Logische bitweise XOR-Operation mit dnum-Daten

Verwendung

`BitXOrDnum` (*Bit eXclusive Or dnum*) wird verwendet, um eine logische bitweise XOR-Operation mit `dnum`-Datentypen abzuarbeiten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `BitXOrDnum`.

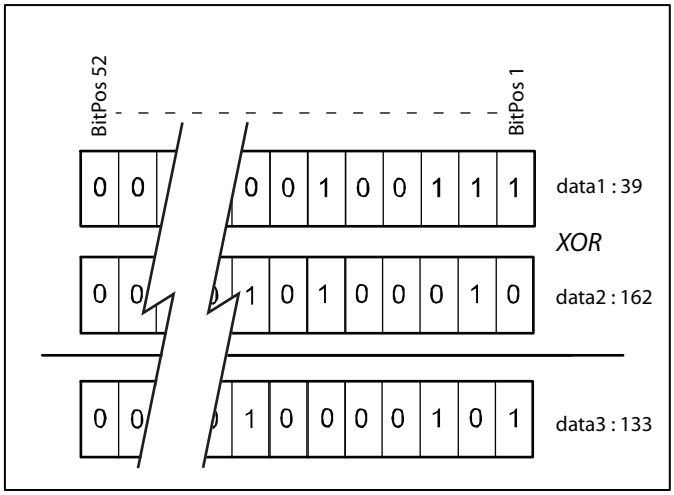
Beispiel 1:

```
VAR dnum data1 := 39;
VAR dnum data2 := 162;
VAR dnum data3;

data3 := BitXOrDnum(data1, data2);
```

Die logische bitweise XOR-Operation wird mit `data1` und `data2` abgearbeitet und das Ergebnis wird an `data3` (ganzzahlige Darstellung) zurückgegeben.

Die folgende Abbildung zeigt eine logische bitweise XOR-Operation.



xx1200000010

Rückgabewert

Datentyp: `dnum`

Das Ergebnis der logischen bitweisen XOR-Operation in ganzzahliger Darstellung.

Argumente

```
BitXOrDnum (Value1 Value2)
```

`Value1`

Datentyp: `dnum`

Der erste Bitdatenwert in ganzzahliger Darstellung.

`Value2`

Datentyp: `dnum`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der zweite Bitdatenwert in ganzzahliger Darstellung.

Einschränkungen

Der Bereich für den Datentyp `dnum` liegt zwischen 0 und 4503599627370495.

Syntax

```
BitXOrDnum '('
  [Value1 ':='] <expression (IN) of dnum> ','
  [Value2 ':='] <expression (IN) of dnum>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Logische bitweise XOR-Operation mit byte-Daten	*BitXOr - Logische bitweise XOR-Operation mit Byte-Daten auf Seite 1214*
Datentyp `dnum`	*dnum – Doppelte numerische Werte auf Seite 1690*
Logische bitweise AND-Operation mit dnum-Daten	*BitAndDnum – Logische bitweise AND-Operation mit dnum-Daten auf Seite 1191*
Logische bitweise OR-Operation mit dnum-Daten	*BitOrDnum – Logische bitweise OR-Operation mit dnum-Daten auf Seite 1208*
Logische bitweise NEGATION-Operation mit dnum-Daten	*BitNegDnum – Logische bitweise NEGATION-Operation mit dnum-Daten auf Seite 1204*
Andere Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.29 ByteToStr - Konvertieren von byte-Daten in string-Daten

Verwendung

`ByteToStr` (*Byte To String*) wird verwendet, um ein Byte in eine Zeichenfolge mit einem definierten Byte-Datenformat zu konvertieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ByteToStr`.

Beispiel 1:

```
VAR string con_data_buffer{5};
VAR byte data1 := 122;

con_data_buffer{1} := ByteToStr(data1);
```

Der Inhalt der Datenfeldkomponente `con_data_buffer{1}` lautet nach der Funktion `ByteToStr ...` "122".

```
con_data_buffer{2} := ByteToStr(data1\Hex);
```

Der Inhalt der Datenfeldkomponente `con_data_buffer{2}` lautet nach der Funktion `ByteToStr ...` "7A".

```
con_data_buffer{3} := ByteToStr(data1\Okt);
```

Der Inhalt der Datenfeldkomponente `con_data_buffer{3}` lautet nach der Funktion `ByteToStr ...` "172".

```
con_data_buffer{4} := ByteToStr(data1\Bin);
```

Der Inhalt der Datenfeldkomponente `con_data_buffer{4}` ist nach der Funktion `ByteToStr ...` „0111101“.

```
con_data_buffer{5} := ByteToStr(data1\Char);
```

Der Inhalt der Datenfeldkomponente `con_data_buffer{5}` lautet nach der Funktion `ByteToStr ...` "z".

Rückgabewert

Datentyp: `string`

Das Ergebnis der Konvertierung mit folgendem Format:

Format	Zeichen	Zeichenfolgenlänge:	Bereich
Dec .....:	’0’ -’ 9’	1-3	"0" - "255"
Hex .....:	’0’ -’ 9’, ’A’ -’F’	2	"00" - "FF"
Oct .....:	’0’ - ’7’	3	"000" - "377"
Bin .....:	’0’ - ’1’	8	"00000000" - "11111111"
Char ....:	Beliebiges ASCII-Zeichen (*)	1	Ein ASCII-Zeichen

(*) Bei nicht schreibbaren ASCII-Zeichen wird ein RAPID-Zeichencodeformat zurückgegeben (z. B. „\07“ für das Steuerzeichen BEL).

Argumente

```
ByteToStr (BitData [\Hex] | [\Okt] | [\Bin] | [\Char])
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`BitData`

Datentyp: `byte`

Die zu konvertierenden Bitdaten.

Wenn das optionale switch-Argument weggelassen wird, werden die Daten in das `decimal`-Format (Dec) konvertiert.

`[\Hex]`

Hexadecimal

Datentyp: `switch`

Die Daten werden in `hexadecimal`-Format konvertiert.

`[\Okt]`

Octal

Datentyp: `switch`

Die Daten werden in `octal`-Format konvertiert.

`[\Bin]`

Binary

Datentyp: `switch`

Die Daten werden in `binary`-Format konvertiert.

`[\Char]`

Character

Datentyp: `switch`

Die Daten werden in `ASCII`-Format.

Einschränkungen

Der Bereich für den Datentyp `byte` ist 0 bis 255 dezimal.

Syntax

```
ByteToStr '('
  [BitData ':='] <expression (IN) of byte>
  ['\' Hex ] | ['\' Okt] | ['\' Bin] | ['\' Char]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Konvertieren von string-Daten in Byte-Daten	*StrToByte - Konvertieren von string-Daten in Byte-Daten auf Seite 1535*
Andere Bit- (Byte-) Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Andere String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.30 CalcJointT - Berechnen der Achsenwinkel vom Roboterziel

Verwendung

`CalcJointT` (*Calculate Joint Target*) wird verwendet, um die Winkel der Roboterachsen und der externen Achsen von einem angegebenen `robtarget` zu berechnen.

Das eingegebene `robtarget` muss in dem Koordinatensystem angegeben werden, das im Argument für `Tool`, `WObj`, in der bei der Abarbeitung aktiven Programmverschiebung (`ProgDisp`) sowie dem Offset der externen Achsen (`EOffs`) angegeben ist. Die zurückgegebenen `jointtarget`-Daten werden im Kalibrierungs-Koordinatensystem angegeben.

Bei einer MultiMove-Anwendung im teilkoordinierten oder synchronisierten, koordinierten Modus mit einem durch eine mechanische Einheit in einen anderen Programmtask bewegten koordinierten Werkobjekt kann die Funktion `CalcJointT` unter folgenden Bedingungen verwendet werden:

- Die aktuelle Position des durch die mechanische Einheit bewegten koordinierten Werkobjekts kann in der Berechnung (aktuelles Anwender-Koordinatensystem) verwendet werden. Alle anderen Daten werden vom RAPID-Programm abgerufen.
- Die mechanische Einheit in einer anderen Programmtask steht still.
- Das Argument `\UseCurWObjPos` wird verwendet.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `CalcJointT`.

Beispiel 1:

```
VAR jointtarget jointpos1;
CONST robtarget p1 := [...];
jointpos1 := CalcJointT(p1, tool1 \WObj:=wobj1);
```

Der `jointtarget` -Wert, der dem `robtarget` -Wert `p1` entspricht, wird in `jointpos1` gespeichert. Das Werkzeug `tool1` und das Werkobjekt `wobj1` werden zur Berechnung der Achsenwinkel `jointpos1` verwendet.

Beispiel 2:

```
VAR jointtarget jointpos2;
CONST robtarget p2 := [...];
jointpos2 := CalcJointT(\UseCurWObjPos, p2, tool2 \WObj:=orb1);
```

Der `jointtarget` -Wert, der dem `robtarget` -Wert `p2` entspricht, wird in `jointpos2` gespeichert. Das Werkzeug `tool2` und das Werkobjekt `orb1` werden zur Berechnung der Achsenwinkel `jointpos2` verwendet. Für die Berechnung wird die aktuelle Position des stillstehenden Manipulators `orb1` herangezogen, der sich jedoch nicht in derselben Programmtask wie der TCP-Roboter befindet.

Beispiel 3

```
VAR jointtarget jointpos3;
CONST robtarget p3 := [...];
VAR errnum myerrnum;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
jointpos3 := CalcJointT(p3, tool2 \WObj:=orb1
     \ErrorNumber:=myerrnum);
IF myerrnum = ERR_ROBLIMIT THEN
  TPWrite "Joint jointpos3 can not be reached.";
  TPWrite "jointpos3.robax.rax_1: "+ValToStr(jointpos3.robax.rax_1);
  ..
  ..
  TPWrite "jointpos3.extax.eax_f"+ValToStr(jointpos3.extax.eax_f);
ELSEIF myerrnum = ERR_OUTSIDE_REACH THEN
  TPWrite "Joint jointpos3 is outside reach.";
  TPWrite "jointpos3.robax.rax_1: "+ValToStr(jointpos3.robax.rax_1);
  ..
  ..
  TPWrite "jointpos3.extax.eax_f"+ValToStr(jointpos3.extax.eax_f);
ELSE
  MoveAbsJ jointpos3, v100, fine, tool2 \WObj:=orb1;
ENDIF
```

Der `jointtarget`-Wert, der dem `robtarget`-Wert `p3` entspricht, wird in `jointpos3` gespeichert. Er wird verwendet, wenn die Position erreicht werden kann, andernfalls wird der `jointtarget`-Wert am FlexPendant angezeigt.

Rückgabewert

Datentyp: `jointtarget`

Winkel in Grad für die Achsen des Roboters an der Seite des Arms.

Werte für die externen Achsen, in mm für lineare Achsen, in Grad für Rotationsachsen.

Die zurückgegebenen Werte beziehen sich immer auf die Kalibrierungsposition.

Argumente

```
CalcJointT ( [\UseCurWObjPos] Rob_target Tool [\WObj]
     [\ErrorNumber])
```

`[\UseCurWObjPos]`

Datentyp: `switch`

Verwenden Sie für die Berechnung die aktuelle Position des durch die mechanische Einheit bewegten koordinierten Werkobjekts (aktuelles Anwender-Koordinatensystem). Alle anderen Daten werden aus dem RAPID-Programm abgerufen.

`Rob_target`

Datentyp: `robtarget`

Die Position des Roboters und der externen Achsen in der äußersten Koordinatensystemebene bezüglich des angegebenen Werkzeugs und Werkobjekts sowie die Programmverschiebung (`ProgDisp`) und/oder der Offset der externen Achsen (`EOffs`), die während der Abarbeitung aktiv sind.

`Tool`

Datentyp: `tooldata`

Werkzeug zur Berechnung der Achsenwinkel des Roboters.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.30 CalcJointT - Berechnen der Achsenwinkel vom Roboterziel
RobotWare - OS
Fortsetzung

`[\WObj]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die Roboterposition bezieht.

Wenn dieses Argument weggelassen wird, wird das Werkobjekt `wobj0` verwendet. Dieses Argument muss angegeben werden, wenn ein stationäres Werkzeug, koordinierte externe Achsen oder ein Conveyor verwendet wird.

`[\ErrorNumber]`

Error number

Datentyp: `errnum`

Eine Variable (`VAR` oder `PERS`), welche die Fehlerkonstante `ERR_ROBLIMIT` enthält, wenn sich mindestens eine Achse außerhalb der Achsgrenzen befindet oder die Grenzen für mindestens eine verbundene Achse überschritten werden, oder `ERR_OUTSIDE_REACH` wenn die Position (robtarget) außerhalb des Arbeitsbereichs des Roboters liegt. Wenn dieses optionale Argument verwendet wird und die Variable nach der Ausführung der Funktion auf `ERR_ROBLIMIT` oder `ERR_OUTSIDE_REACH` gesetzt wird, handelt es sich beim Rückgabewert um einen `jointtarget`-Wert entsprechend dem verwendeten `robtarget`.

Beim Weglassen dieser optionalen Variable wird die Fehlerbehandlung abgearbeitet und der zurückgegebene `jointarget`-Wert wird nicht aktualisiert, wenn sich eine Achse außerhalb des Arbeitsbereichs befindet oder die Grenzen überschritten wurden.

Programmabarbeitung

Das ausgegebene `jointtarget` wird von der Eingabe `robtarget` berechnet. Wenn das Argument `\UseCurWObjPos` verwendet wird, dann kommt die verwendete Position von der aktuellen Position der mechanischen Einheit, die den Benutzerrahmen steuert. Um die Gelenkwinkel des Roboters zu berechnen, werden der angegebene `Tool`, `WObj` (einschließlich Benutzerkoordinatensystem) und `ProgDisp`, das zum Abarbeitungszeitpunkt aktiv ist, berücksichtigt. Für die Berechnung der externen Achsenposition während der Abarbeitung werden aktive `EOffs` ebenfalls berücksichtigt.

Die Berechnung wählt stets die Roboterkonfiguration gemäß den angegebenen Konfigurationsdaten in den übergebenen `robtarget`-Daten. Die Instruktionen `ConfL` und `ConfJ` beeinflussen dieses Berechnungsprinzip nicht. Bei Verwendung der Handgelenksingularität wird Roboterachse 4 auf 0 Grad gesetzt.

Bei einer aktiven Programmverschiebung (`ProgDisp`) und/oder Offset von externen Achsen (`EOffs`) während des Speichern des Roboterziels müssen dieselbe Programmverschiebung und/oder derselbe Offset der externen Achsen aktiv sein, wenn `CalcJointT` abgearbeitet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_ROBLIMIT	Die Position ist erreichbar, aber mindestens eine Achse liegt außerhalb der Achsenbeschränkungen oder die Grenzen für mindestens eine verbundene Achse werden überschritten.
ERR_OUTSIDE_REACH	Die Position (robtarget) befindet sich außerhalb des Arbeitsbereichs des Roboters.
ERR_WOBJ_MOVING	Die mechanische Einheit, die das Werkobjekt steuert (Anwender-Koordinatensystem), steht während der Abarbeitung von CalJointT \UseCurWobjPos nicht still.

Einschränkung

Bei Verwendung eines Koordinatensystems muss die koordinierte Einheit aktiviert werden, bevor CalcJointT verwendet wird.

Die mechanische Einheit, die das Anwender-Koordinatensystem im Werkobjekt steuert, muss normalerweise in derselben Programmtask wie der TCP-Roboter verfügbar sein, der CalcJointT abarbeitet.

Normalerweise berechnet CalcJointT mit robtarget, tooldata und wobjdata aus dem RAPID-Programm jointtarget. Für koordinierte Werkobjekte wird die Position der mechanischen Einheit als Position der externen Achsen im robtarget angegeben. Dies ist nicht der Fall, wenn die mechanische Einheit von einer anderen Programmtask gesteuert wird (MultiMove-System) oder wenn die mechanische Einheit nicht vom Steuerungssystem gesteuert wird (Conveyor). Für das MultiMove-System, doch nicht für den Conveyor, kann das Argument \UseCurWObjPos verwendet werden, wenn die mechanische Einheit während der Abarbeitung von CalCJointT stillsteht.

Syntax

```
CalcJointT '('
  [ '\'UseCurWObjPos ','']
  [ Rob_target ':='] <expression (IN) of robtarget> ','
  [ Tool ':=' ] <persistent (PERS) of tooldata>
  [ '\' WObj ':=' <persistent (PERS) of wobjdata>]
  [ '\' ErrorNumber ':=' <variable or persistent (INOUT) of errnum>]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs jointtarget.

Weitere Informationen

Information über	Siehe
Berechnen des Roboterziels aus dem Achsenziel	*CalcRobT - Berechnen des Roboterziels aus dem Achsenziel auf Seite 1225*
Definition der Position	*robtarget - Positionsdaten auf Seite 1778*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.30 CalcJointT - Berechnen der Achsenwinkel vom Roboterziel

RobotWare - OS

Fortsetzung

Information über	Siehe
Definition der Achsenposition	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Programmverschiebungs-Koordinatensystem	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Koordinatensystem für externen Achsen-Offset	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.31 CalcRobT - Berechnen des Roboterziels aus dem Achsenziel

Verwendung

`CalcRobT` (*Calculate Robot Target*) wird zum Berechnen eines `robtarget` anhand eines angegebenen `jointtarget` verwendet.

Diese Funktion gibt einen Wert des Typs `robtarget` mit Position (x, y, z), Orientierung (q1 ... q4), Roboterachsenkonfiguration und Position der externen Achsen zurück.

Die übergebenen `jointtarget`-Daten werden im Kalibrierungs-Koordinatensystem ausgedrückt.

Die zurückgegebenen `robtarget`-Daten werden im Kalibrierungs-Koordinatensystem ausgedrückt. Das angegebene Werkzeug, das angegebene Werkobjekt und die bei der Abarbeitung aktive Programmverschiebung (`ProgDisp`) sowie der Offset der externen Achsen (`EOffs`) werden berücksichtigt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CalcRobT`.

Beispiel 1:

```
VAR robtarget p1;
CONST jointtarget jointpos1 := [...];

p1 := CalcRobT(jointpos1, tool1 \WObj:=wobj1);
```

Der `robtarget`-Wert, der dem `jointtarget`-Wert `jointpos1` entspricht, wird in `p1` gespeichert. Das Werkzeug `tool1` und das Werkobjekt `wobj1` werden zur Berechnung der Position `p1` verwendet.

Rückgabewert

Datentyp: `robtarget`

Die Position von Roboter und externen Achsen werden im Datentyp `robtarget` zurückgegeben und im äußersten Koordinatensystem ausgedrückt. Das angegebene Werkzeug, das angegebene Werkobjekt und die bei der Abarbeitung aktive Programmverschiebung (`ProgDisp`) sowie der Offset der externen Achsen (`EOffs`) werden berücksichtigt.

Ohne aktive `ProgDisp` wird die Roboterposition im Objekt-Koordinatensystem ausgedrückt. Ohne aktive `EOffs` wird die Position der externen Achsen im Kalibrierungs-Koordinatensystem ausgedrückt.

Argumente

```
CalcRobT( Joint_target Tool [\WObj] )
```

`Joint_target`

Datentyp: `jointtarget`

Die Achsenposition für die Roboterachsen und die externen Achsen in Bezug auf das Kalibrierungs-Koordinatensystem.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.31 CalcRobT - Berechnen des Roboterziels aus dem Achsenziel
RobotWare - OS
Fortsetzung

`Tool`

Datentyp: `tooldata`

Werkzeug zur Berechnung der Roboterposition.

`[\WObj]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die von der Funktion zurückgegebene Roboterposition bezieht.

Wenn dieses Argument weggelassen wird, wird das Werkobjekt `wobj0` verwendet. Dieses Argument muss angegeben werden, wenn ein stationäres Werkzeug, koordinierte externe Achsen oder ein Conveyor verwendet wird.

Programmabarbeitung

Das zurückgegebene Achsenziel `robtarget` wird aus dem eingegebenen `jointtarget` berechnet. Um die kartesische Roboterposition zu berechnen, werden das angegebene `ToolWObj` (einschließlich des koordinierten Anwender-Koordinatensystems) und die während der Programmabarbeitung aktive Programmverschiebung `ProgDisp` berücksichtigt.

Für die Berechnung der externen Achsenposition während der Abarbeitung werden aktive `EOffs` ebenfalls berücksichtigt.

Einschränkung

Bei Verwendung eines Koordinatensystems muss die koordinierte Einheit aktiviert werden, bevor `CalcRobT` verwendet wird. Die koordinierte Einheit muss sich auch in derselben Task wie der Roboter befinden.

Syntax

```
CalcRobT '('
  [Joint_target ':=' ] <expression (IN) of jointtarget> ','
  [Tool ':=' ] <persistent (PERS) of tooldata>
  ['\' WObj ':=' <persistent (PERS) of wobjdata>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `robtarget`.

Weitere Informationen

Information über	Siehe
Berechnen des Achsenziels aus dem Roboterziel	*CalcJointT - Berechnen der Achsenwinkel vom Roboterziel auf Seite 1220*
Definition der Position	*robtarget - Positionsdaten auf Seite 1778*
Definition der Achsenposition	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Programmverschiebungs-Koordinatensystem	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Koordinatensystem für externen Achsen-Offset	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

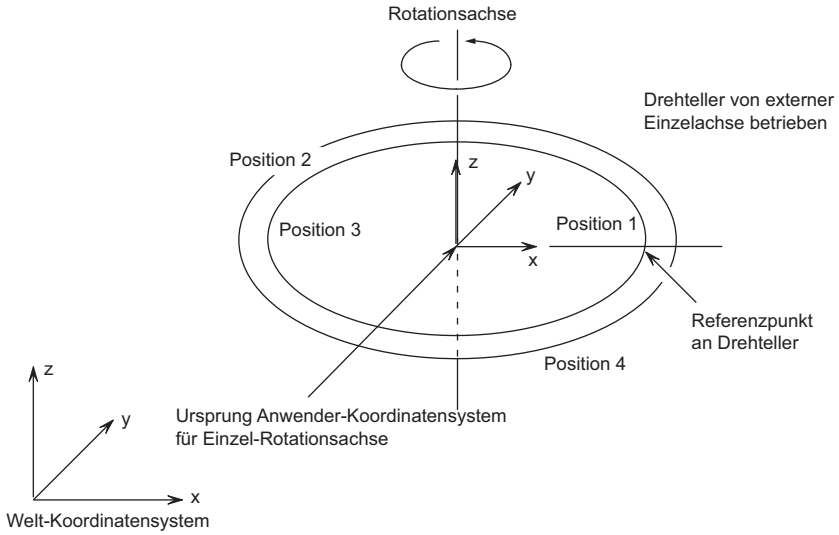
2.32 CalcRotAxFrameZ – Berechnen eines Rotationsachsenrahmens

Verwendung

`CalcRotAxFrameZ` (*Calculate Rotational Axis Frame with positive Z-point*) wird verwendet, um das Benutzer-Koordinatensystem für eine mechanische Rotationsachseneinheit zu berechnen. Diese Funktion kann verwendet werden, wenn sich der Masterroboter und die zusätzliche Achse in verschiedenen RAPID-Tasks befinden. Wenn sie sich in derselben Task befinden, sollte die Funktion `CalcRotAxisFrame` verwendet werden.

Beschreibung

Die Definition eines Anwender-Koordinatensystems für eine externe Rotationsachse erfordert, dass der Drehteller (oder eine ähnliche mechanische Struktur) auf der externen Achse über einen markierten Referenzpunkt verfügt. Darüber hinaus müssen das Basis-Koordinatensystem und der TCP des TCP-Roboters kalibriert werden. Die Kalibrierungsprozedur besteht aus einer Reihe von Positionierungen für den TCP des Roboters auf dem Referenzpunkt, während der Drehteller in verschiedene Winkel gedreht wird Zusätzlich muss der TCP des Roboters in positive z-Richtung positioniert werden. Informationen zur Definition von Punkten für eine Rotationsachse finden Sie in der folgenden Abbildung.



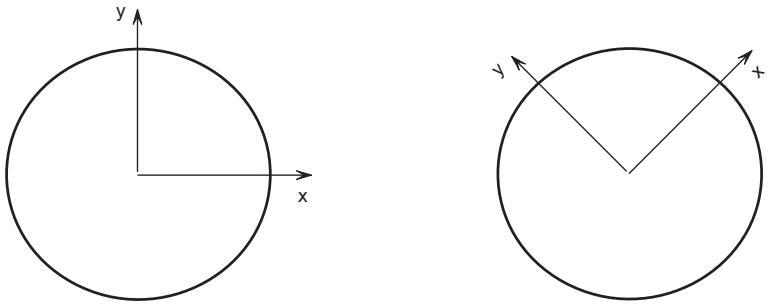
xx0500002468

Der Ursprung des Anwender-Koordinatensystems für die Rotationsachse liegt in der Mitte des Drehtellers. Die z-Richtung stimmt mit der Rotationsachse überein und die x-Achse verläuft durch den Referenzpunkt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Nachstehende Abbildung zeigt das Anwender-Koordinatensystem für zwei verschiedene Positionierungen des Drehtellers (Drehteller von oben gesehen).



xx0500002469

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CalcRotAxFrameZ`.

Beispiel 1:

```
CONST robtarget pos1 := [...];
CONST robtarget pos2 := [...];
CONST robtarget pos3 := [...];
CONST robtarget pos4 := [...];
CONST robtarget zpos;
VAR robtarget targetlist{10};
VAR num max_err := 0;
VAR num mean_err := 0;
VAR pose resFr:=[...];
PERS tooldata tMyTool:= [...];

! Instructions for creating/ModPos pos1 - pos4 with TCP pointing
     at the turntable.
MoveJ pos1, v10, fine, tMyTool;
MoveJ pos2, v10, fine, tMyTool;
MoveJ pos3, v10, fine, tMyTool;
MoveJ pos4, v10, fine, tMyTool;

!Instruction for creating/ModPos zpos with TCP pointing at a point
     in positive z direction
MoveJ zpos, v10, fine, tMyTool;

! Add the targets to the array
targetlist{1}:= pos1;
targetlist{2}:= pos2;
targetlist{3}:= pos3;
targetlist{4}:= pos4;

resFr:=CalcRotAxFrameZ(targetlist, 4, zpos, max_err, mean_err);

! Update the system parameters.
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.32 CalcRotAxFrameZ – Berechnen eines Rotationsachsenrahmens

RobotWare - OS

Fortsetzung

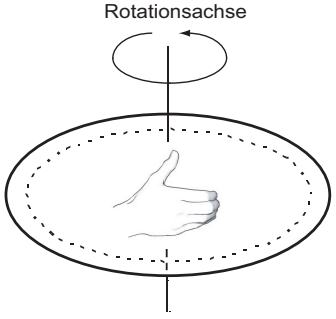
```
IF (max_err < 1.0) AND (mean_err < 0.5) THEN
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_x",
       resFr.trans.x/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_y",
       resFr.trans.y/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_z",
       resFr.trans.z/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u0",
       resFr.rot.q1;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u1",
       resFr.rot.q2;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u2",
       resFr.rot.q3;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u3",
       resFr.rot.q4;
  TPReadFK reg1,"Warmstart required for calibration to take
       effect.", stEmpty, stEmpty, stEmpty, stEmpty,"OK";
  WarmStart;
ENDIF
```

Die vier Positionen `pos1` – `pos4` werden so erstellt/positioniert, dass das Roboterwerkzeug `tMyTool` auf denselben Referenzpunkt auf der externen Achse `STN_1`, doch mit unterschiedlicher Achsendrehung gerichtet ist. Die Position, `zpos`, wird so erstellt/positioniert, dass das Roboterwerkzeug `tMyTool` in die positive z-Richtung zeigt, gemäß der Definition der positiven z-Richtung einer externen rotierenden mechanischen Einheit. Zur Verwendung der Definition der positiven z-Richtung einer externen rotierenden mechanischen Einheit siehe *Beschreibung auf Seite 1228*. Die Punkte werden dann zur Berechnung des Basis-Koordinatensystems der externen Achse, `resFr`, in Bezug auf das Welt-Koordinatensystem verwendet. Schließlich wird das Koordinatensystem in die Konfigurationsdatei geschrieben, und es wird eine `WarmStart`-Instruktion ausgeführt, damit die Änderung wirksam wird.

Hinweis

Definition der positiven z-Richtung einer externen rotierenden mechanischen Einheit.

Richten Sie die Finger der rechten Hand exakt an der positiven Drehachse der Rotationsachse aus. Die Richtung des Daumens definiert dann die positive z-Richtung. Siehe Abbildung unten.



xx0500002472

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Rückgabewert

Datentyp: `pose`

Das berechnete Koordinatensystem.

Argumente

```
CalcRotAxFrameZ (TargetList TargetsInList PositiveZPoint
    MaxErrMeanErr)
```

`TargetList`

Datentyp: `robtarget`

Datenfeld mit Roboterzielen, das die Positionen enthält, die durch Punkte am Drehteller definiert werden. Die Mindestanzahl der Roboterziele ist 4, die Höchstanzahl 10.

`TargetsInList`

Datentyp: `num`

Anzahl der Roboterziele in einem Datenfeld.

`PositiveZPoint`

Datentyp: `robtarget`

`robtarget` enthält die Position, die durch die Angabe eines Punkts in positiver z-Richtung definiert wurde. Dazu ist die Definition der positiven z-Richtung einer externen rotierenden mechanischen Einheit zu verwenden (siehe *Beschreibung auf Seite 1228*.

`MaxErr`

Maximum Error

Datentyp: `num`

Geschätzter maximaler Fehler in mm.

`MeanErr`

Mean Error

Datentyp: `num`

Geschätzter mittlerer Fehler in mm.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FRAME`	Die Positionen weisen nicht die erforderliche Relation auf oder sie sind nicht genau genug angegeben.

Syntax

```
CalcRotAxFrameZ '('
  [TargetList ':='] <array {*} (IN) of robtarget> ','
  [TargetsInList ':='] <expression (IN) of num> ','
  [PositiveZPoint ':='] <expression (IN) of robtarget> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.32 CalcRotAxFrameZ – Berechnen eines Rotationsachsenrahmens

RobotWare - OS

Fortsetzung

```
[MaxErr ':='] <variable (VAR) of num> ','
[MeanErr ':='] <variable (VAR) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

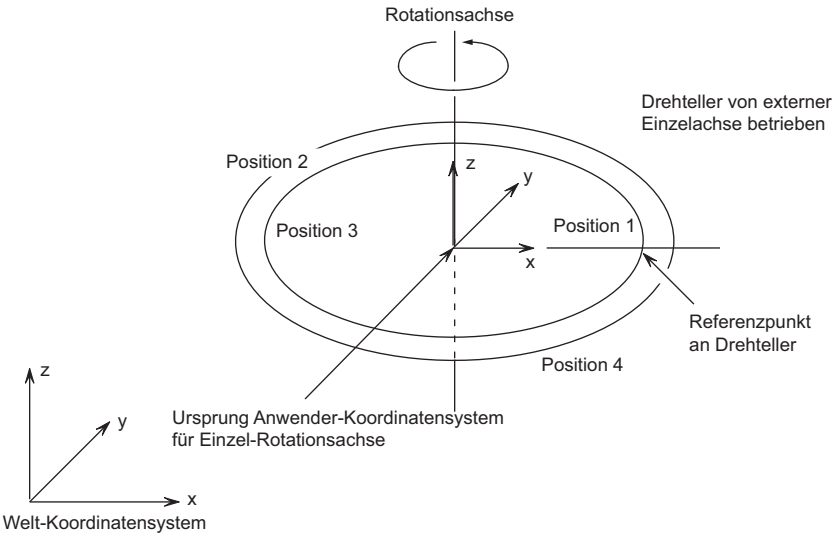
2.33 CalcRotAxisFrame – Berechnen eines Rotationsachsenrahmens

Verwendung

`CalcRotAxisFrame` (*Calculate Rotational Axis Frame*) wird verwendet, um das Benutzer-Koordinatensystem für eine mechanische Rotationsachseneinheit zu berechnen. Diese Funktion kann verwendet werden, wenn sich der Masterroboter und die zusätzliche Achse in derselben RAPID-Task befinden. Wenn sie sich in verschiedenen Tasks befinden, sollte die Funktion `CalcRotAxFrameZ` verwendet werden.

Beschreibung

Die Definition eines Anwender-Koordinatensystems für eine externe Rotationsachse erfordert, dass der Drehteller (oder eine ähnliche mechanische Struktur) auf der externen Achse über einen markierten Referenzpunkt verfügt. Darüber hinaus müssen das Basis-Koordinatensystem und der TCP des Roboters kalibriert werden. Die Kalibrierungsprozedur besteht aus einer Reihe von Positionierungen für den TCP des Roboters auf dem Referenzpunkt, während der Drehteller in verschiedene Winkel gedreht wird Informationen zur Definition von Punkten für eine Rotationsachse finden Sie in der folgenden Abbildung.



xx0500002468

Der Ursprung des Anwender-Koordinatensystems für die Rotationsachse liegt in der Mitte des Drehtellers. Die z-Richtung stimmt mit der Rotationsachse überein und die x-Achse verläuft durch den Referenzpunkt.

Fortsetzung auf nächster Seite

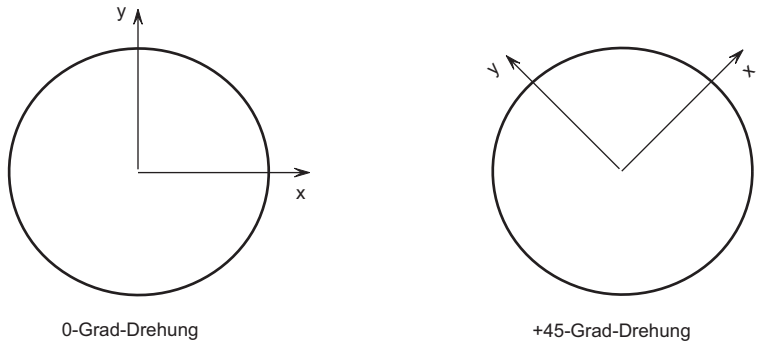
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.33 CalcRotAxisFrame – Berechnen eines Rotationsachsenrahmens

RobotWare - OS

Fortsetzung

Nachstehende Abbildung zeigt das Anwender-Koordinatensystem für zwei verschiedene Positionierungen des Drehtellers (Drehteller von oben gesehen).



xx0500002469

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CalcRotAxisFrame`.

Beispiel 1:

```
CONST robtarget pos1 := [...];
CONST robtarget pos2 := [...];
CONST robtarget pos3 := [...];
CONST robtarget pos4 := [...];
VAR robtarget targetlist{10};
VAR num max_err := 0;
VAR num mean_err := 0;
VAR pose resFr:=[...];
PERS tooldata tMyTool:= [...];

! Instructions needed for creating/ModPos pos1 - pos4 with TCP
     pointing at the turntable.
MoveJ pos1, v10, fine, tMyTool;
MoveJ pos2, v10, fine, tMyTool;
MoveJ pos3, v10, fine, tMyTool;
MoveJ pos4, v10, fine, tMyTool;

! Add the targets to the array
targetlist{1}:= pos1;
targetlist{2}:= pos2;
targetlist{3}:= pos3;
targetlist{4}:= pos4;

resFr:=CalcRotAxisFrame(STN_1 , targetlist, 4, max_err, mean_err);

! Update the system parameters.
IF (max_err < 1.0) AND (mean_err < 0.5) THEN
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_x",
       resFr.trans.x/1000;
  WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_y",
       resFr.trans.y/1000;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_pos_z",
         resFr.trans.z/1000;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u0",
         resFr.rot.q1;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u1",
         resFr.rot.q2;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u2",
         resFr.rot.q3;
    WriteCfgData "/MOC/SINGLE/STN_1", "base_frame_orient_u3",
         resFr.rot.q4;
    TPReadFK reg1,"Warmstart required for calibration to take
         effect.", stEmpty, stEmpty, stEmpty, stEmpty, "OK";
    WarmStart;
  ENDIF
```

Die vier Positionen `pos1` – `pos4` werden so erstellt/positioniert, dass das Roboterwerkzeug `tMyTool` auf denselben Referenzpunkt auf der externen Achse `STN_1`, doch mit unterschiedlicher Achsendrehung gerichtet ist. Die Punkte werden dann zur Berechnung des Basis-Koordinatensystems der externen Achse, `resFr`, in Bezug auf das Welt-Koordinatensystem verwendet. Schließlich wird das Koordinatensystem in die Konfigurationsdatei geschrieben, und es wird eine `WarmStart`-Instruktion ausgeführt, damit die Änderung wirksam wird.

Rückgabewert

Datentyp: `pose`

Das berechnete Koordinatensystem.

Argumente

```
CalcRotAxisFrame (MechUnit [\AxisNo] TargetList TargetsInList MaxErr
     MeanErr)
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, die kalibriert werden soll.

`[\AxisNo]`

Datentyp: `num`

Optionales Argument, das die Achsennummer definiert, für die ein Koordinatensystem bestimmt werden soll. Standardwert ist 1 für eine einzelne Rotationsachse. Für mechanische Einheiten mit mehreren Achsen muss die Achsennummer mit diesem Argument angegeben werden.

`TargetList`

Datentyp: `robtarget`

Datenfeld mit Roboterzielen, das die Positionen enthält, die durch Punkte am Drehteller definiert werden. Die Mindestanzahl der Roboterziele ist 4, die Höchstanzahl 10.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.33 CalcRotAxisFrame – Berechnen eines Rotationsachsenrahmens
RobotWare - OS
Fortsetzung

`TargetsInList`

Datentyp: `num`

Anzahl der Roboterziele in einem Datenfeld.

`MaxErr`

Maximum Error

Datentyp: `num`

Geschätzter maximaler Fehler in mm.

`MeanErr`

Mean Error

Datentyp: `num`

Geschätzter mittlerer Fehler in mm.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FRAME`	Die Positionen weisen nicht die erforderliche Relation auf oder sie sind nicht genau genug angegeben.

Syntax

```
CalcRotAxisFrame '('
  [MechUnit ':='] <variable (VAR) of mecunit>
  [\AxisNo ':=' <expression (IN) of num> ] ','
  [TargetList ':='] <array {*} (IN) of robtarget> ','
  [TargetsInList ':='] <expression (IN) of num> ','
  [MaxErr ':='] <variable (VAR) of num> ','
  [MeanErr ':='] <variable (VAR) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.34 CamGetExposure – Abrufen der kameraspezifischen Daten

Verwendung

`CamGetExposure(Camera Get Exposure)` ist eine Funktion, die die aktuellen Einstellungen einer Kamera liest. Mit dieser Funktion und der Instruktion`CamSetExposure` ist es möglich, die Kameraaufnahmen an die Bedingungen zur Laufzeit anzupassen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CamGetExposure`.

Beispiel 1:

```
VAR num exposuretime;
...
exposuretime:=CamGetExposure(mycamera \ExposureTime);
IF exposuretime = 10 THEN
  CamSetExposure mycamera \ExposureTime:=9.5;
ENDIF
```

Weisen Sie die Kamera `mycamera` an, die Belichtungszeit auf 9,5 ms zu ändern, falls die aktuelle Einstellung bei 10 ms liegt.

Rückgabewert

Datentyp: `num`

Eine der Einstellungen „Belichtungszeit“, „Helligkeit“ oder „Kontrast“ wird von der Kamera als numerischer Wert zurückgegeben.

Argumente

```
CamGetExposure (Camera [\ExposureTime] | [\Brightness] |
     [\Contrast])
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`[\ExposureTime]`

Datentyp: `num`

Gibt die Belichtungszeit der Kamera zurück. Der Wert wird in Millisekunden (ms) angegeben.

`[\Brightness]`

Datentyp: `num`

Gibt die Helligkeitseinstellung der Kamera zurück

`[\Contrast]`

Datentyp: `num`

Gibt die Kontrasteinstellung der Kamera zurück

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.34 CamGetExposure – Abrufen der kameraspezifischen Daten

Integrated Vision
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CAM_BUSY	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
ERR_CAM_COM_TIMEOUT	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden.

Syntax

```
CamGetExposure '('
  [ Camera ':=' ] < variable (VAR) of cameradev >
  ['\'ExposureTime]
  | ['\'Brightness]
  | ['\'Contrast] ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.35 CamGetLoadedJob – Abrufen des Namens der geladenen Kameratask

Verwendung

`CamGetLoadedJob` *(Kamera Abrufen Geladener Job)* ist eine Funktion, die den Namen des aktuell geladenen Jobs von der Kamera liest und ihn in einer Zeichenfolge zurückgibt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CamGetLoadedJob`.

Beispiel 1:

```
VAR string currentjob;
...
currentjob:=CamGetLoadedJob(mycamera);
IF CurrentJob = "" THEN
  TPWrite "No job loaded in camera "+CamGetName(mycamera);
ELSE
  TPWrite "Job "+CurrentJob+" is loaded in camera "
       "+CamGetName(mycamera);
ENDIF
```

Schreiben Sie den Namen des geladenen Auftrags auf das FlexPendant.

Rückgabewert

Datentyp: `string`

Der Name des aktuell geladenen Auftrags für die spezifische Kamera.

Argumente

```
CamGetLoadedJob (Camera)
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Programmabarbeitung

Die Funktion `CamGetLoadedJob` ruft den Namen des aktuell geladenen Auftrags von der Kamera ab. Wenn kein Auftrag auf die Kamera geladen ist, wird eine leere Zeichenfolge zurückgegeben.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_CAM_BUSY`	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
`ERR_CAM_COM_TIMEOUT`	Kommunikationsfehler mit Kamera. Die Kamera ist vermutlich nicht verbunden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.35 CamGetLoadedJob – Abrufen des Namens der geladenen Kameratask

Integrated Vision

Fortsetzung

Name	Fehlerursache
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden.

Syntax

```
CamGetLoadedJob '('
  [ Camera ':=' ] < variable (VAR) of cameradev > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.36 CamGetMode - Aktuellen Modus der Kamera abrufen

Verwendung

`CamGetMode` gibt den aktuellen Modus der Kamera zurück.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `CamGetMode`.

Beispiel 1:

```
VAR camerastatus curr_camerastatus;
...
curr_camerastatus:=CamGetMode(mycamera);
IF curr_camerastatus = CAMERA_DISCON THEN
  TPWrite "The camera is disconnected. Check cabling for camera
       "+CamGetName(mycamera);
ENDIF
```

Aktuellen Modus der Kamera abrufen. Wenn die Kamera getrennt ist, schreiben Sie das auf das FlexPendant.

Rückgabewert

Datentyp: `camerastatus`

Der aktuelle Status der Kamera.

Nur die vordefinierten, symbolischen Konstanten des Typs `camerastatus` können zur Überprüfung des Status verwendet werden.

Argumente

```
CamGetMode( Camera )
```

`Camera`

Datentyp: `cameradev`

Die Kamera, deren Status erfasst werden soll.

Programmabarbeitung

Die Funktion gibt einen der folgenden vordefinierten Kamerastatus zurück:

Syntax

```
CamGetMode '('
  [Camera ':='] <variable (VAR) of cameradev>')'
```

Eine Funktion mit einem Rückgabewert des Typs `camerastatus`.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*
Datentyp `camerastatus`	*camerastatus - Kamera-Kommunikationsstatus auf Seite 1646*
Datentyp `cameradev`	*cameradev – Kameragerät auf Seite 1645*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.37 CamGetName – Abrufen des Namens der verwendeten Kamera

Verwendung

`CamGetName(Camera Get Name)` wird verwendet, um den konfigurierten Namen der Kamera abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CamGetName`.

Beispiel 1:

```
  ...
  logcameraname camera1;
  CamReqImage camera1;
  ...
  logcameraname camera2;
  CamReqImage camera2;
  ...
PROC logcameraname(VAR cameradev camdev)
  TPWrite "Now using camera: "+CamGetName(camdev);
ENDPROC
```

Der Vorgang protokolliert den Namen der aktuell verwendeten Kamera in FlexPendant.

Rückgabewert

Datentyp: `string`

Der Namen der aktuell verwendeten Kamera wird als Zeichenfolge zurückgegeben.

Argumente

```
CamGetName(Camera)
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

Syntax

```
CamGetName( '('
  [ Camera ':=' ] < variable (VAR) of cameradev > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.38 CamNumberOfResults – Abrufen der Anzahl der verfügbaren Ergebnisse

Verwendung

`CamNumberOfResults` (`Camera Number of Results`) ist eine Funktion, die die Anzahl der verfügbaren Bildergebnisse anzeigt und sie als numerischen Wert zurückgibt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CamNumberOfResults`.

Beispiel 1:

```
VAR num foundparts;
...
CamReqImage mycamera;
WaitTime 1;
FoundParts := CamNumberOfResults(mycamera);
TPWrite "Number of identified parts in the camera image:
     "\Num:=foundparts;
```

Ein Bild erfassen. Warten Sie darauf, dass die Bildverarbeitung beendet ist, in diesem Fall 1 Sekunde. Lesen Sie die Nummer der identifizierten Teile und schreiben Sie sie in das FlexPendant.

Rückgabewert

Datentyp: `num`

Gibt die Anzahl der Ergebnisse in der Sammlung für die spezifische Kamera zurück.

Argumente

```
CamNumberOfResults (Camera [\SceneId])
```

`Camera`

Datentyp: `cameradev`

Name der Kamera

`[\SceneId]`

Szenenidentifikation

Datentyp: `num`

`SceneId` ist ein Bezeichner, der angibt, von welchem Bild die Nummer der identifizierten Teile zu lesen ist.

Programmabarbeitung

`CamNumberOfResults` ist eine Funktion, die die Anzahl der verfügbaren Bildergebnisse anzeigt und sie als numerischen Wert zurückgibt. Sie kann verwendet werden, um durch alle verfügbaren Ergebnisse zu schleifen.

Die Funktion gibt die Queue-Ebene direkt zurück, sobald die Funktion ausgeführt wird. Wenn die Funktion direkt nach dem Anfordern des Bilds ausgeführt wird, ist das Ergebnis häufig 0, da die Kamera die Verarbeitung des Bilds noch nicht beendet hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.38 CamNumberOfResults – Abrufen der Anzahl der verfügbaren Ergebnisse
Integrated Vision
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird auf folgenden Wert gesetzt:

Name	Fehlerursache
ERR_CAM_BUSY	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.

Syntax

```
CamNumberOfResults '('
  [ Camera ':=' ] < variable (VAR) of cameradev >
  [ '\'SceneId ':=' < expression (IN) of num > ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.39 CapGetFailSigs - Ausgefallene E/A-Signale abrufen

Verwendung

`CapGetFailSigs` wird verwendet, um die Namen des Signals oder der Signale auszugeben, die während der Überwachung von `CapL` oder `CapC` ausgefallen sind.

Wenn die Überwachung eines oder mehrerer Signale während des Prozesses fehlschlägt, wird ein behebbarer Fehler von der Instruktion `CapL`/`CapC` ausgegeben. Um zu bestimmen, welches Signal oder welche Signale ausgefallen sind, kann in der Fehlerbehandlung `CapGetFailSigs` für alle Fälle von Überwachungsfehlern verwendet werden.

Grundlegendes Beispiel

```
Stringcopied := CapGetFailSigs(Failstring);
```

`Stringcopied` wird der Wert TRUE zugewiesen, wenn das Kopieren erfolgreich war, und FALSE, wenn es fehlgeschlagen ist.

`Failstring` enthält die Signale, die als Text fehlgeschlagen sind. Wenn keine Zeichenfolge kopiert werden konnte, wird die Zeichenfolge `EMPTY` ausgegeben.

Rückgabewert

Datentyp: `bool`

TRUE oder FALSE hängt davon ab, ob die fehlerhafte Zeichenkette geändert wird.

Argumente

```
CapGetFailSigs (ErrorNames)
```

ErrorNames

Datentyp: `string`

`CapGetFailSigs` erfordert eine String-Variable als Eingabeparameter.

Einschränkungen

Wenn viele Signale in einer Überwachungsliste gleichzeitig ausgefallen sind, werden nur drei von ihnen mit `CapGetFailSigs` gemeldet.

Syntax

```
CapGetFailSigs '('
[ErrorNames ':='] < variable (INOUT) of string >')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Continuous Application Platform	*Application manual - Continuous Application Platform*
Instruktion `InitSuperv`	*InitSuperv - Zurücksetzen der gesamten CAP-Überwachung auf Seite 284*
Instruktion `SetupSuperv`	*SetupSuperv - Einrichtungsbedingungen für Signalüberwachung in CAP auf Seite 725*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.39 CapGetFailSigs - Ausgefallene E/A-Signale abrufen

Continuous Application Platform (CAP)

Fortsetzung

Information über	Siehe
Instruktion `RemoveSuperv`	*RemoveSuperv Bedingung für ein Signal entfernen auf Seite 607*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.40 CDate - Lesen des aktuellen Datums als Zeichenfolge

Verwendung

`CDate` (*Current Date*) wird verwendet, um das aktuelle Systemdatum zu lesen.

Mithilfe dieser Funktion kann das aktuelle Datum am FlexPendant angezeigt oder in eine Textdatei eingefügt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CDate`.

Siehe auch *Weitere Beispiele auf Seite 1247*.

Beispiel 1:

```
VAR string date;
date := CDate();
```

Das aktuelle Datum wird in der Variablen `date` gespeichert.

Rückgabewert

Datentyp: `string`

Das aktuelle Datum in einer Zeichenfolge.

Das Standarddatum ist im Format „Jahr-Monat-Tag“, z. B. 1998-01-29.

Weitere Beispiele

Weitere Beispiele für die Funktion `CDate` sind unten aufgeführt.

Beispiel 1:

```
VAR string date;
date := CDate();
TPWrite "The current date is: "+date;
Write logfile, date;
```

Das aktuelle Datum wird am FlexPendant angezeigt und in eine Textdatei geschrieben.

Syntax

```
CDate '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Zeitinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Einstellen der Systemuhr	*Bedienungsanleitung - IRC5 mit FlexPendant*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.41 CJointT - Lesen der aktuellen Achsenwinkel

Verwendung

`CJointT` (*Current Joint Target*) wird verwendet, um die aktuellen Winkel der Roboterachsen und der externen Achsen abzulesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CJointT`.

Siehe auch *Weitere Beispiele auf Seite 1249*.

Beispiel 1:

```
VAR jointtarget joints;
joints := CJointT();
```

Die aktuellen Winkel der Roboterachsen und der externen Achsen werden in `joints` gespeichert.

Rückgabewert

Datentyp: `jointtarget`

Die aktuellen Winkel in Grad für die Achsen des Roboters an der Seite des Arms.

Die aktuellen Werte für die externen Achsen, in mm für lineare Achsen, in Grad für Rotationsachsen.

Die zurückgegebenen Werte beziehen sich auf die Kalibrierungsposition.

Argumente

```
CJointT ([\TaskRef]|[\TaskName])
```

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, aus der das `jointtarget` gelesen werden soll.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“ z. B. für den Task `T_ROB1`, und die Variablenidentität lautet `T_ROB1Id`.

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, aus der das `jointtarget` gelesen werden soll.

Wenn keines der Argumente `\TaskRef` oder `\TaskName` angegeben ist, wird die aktuelle Task verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.
ERR_NOT_MOVETASK	Argument \TaskRef oder \TaskName gibt eine Nicht-Bewegungstask an.

Es wird jedoch kein Fehler generiert, wenn das Argument \TaskRef oder \TaskName die Nicht-Bewegungstask angibt, die die Funktion CJointT ausführt (Verweis auf die eigene Nicht-Bewegungstask). Die Position wird dann aus der verbundenen Bewegungstask abgerufen.

Weitere Beispiele

Weitere Beispiele für die Funktion CJointT sind unten aufgeführt.

Beispiel 1:

```
! In task T_ROB1
VAR jointtarget joints;
joints := CJointT(\TaskRef:=T_ROB2Id);
```

Die aktuellen Positionen der Roboterachse und der externen Achsen in der Task T_ROB2 werden in joints in Task T_ROB1 gespeichert.

Beachten Sie, dass der Roboter im Task T_ROB2 sich eventuelle bewegt, wenn die Position berechnet wird. Um sicherzustellen dass der Roboter still steht, kann ein Stopppunkt fine in der vorherigen Bewegungsinstruktion in Task T_ROB2 programmiert werden und mit der Instruktion WaitSyncTask können die Instruktionen in Task T_ROB1 synchronisiert werden.

Beispiel 2:

```
! In task T_ROB1
VAR jointtarget joints;
joints := CJointT(\TaskName:="T_ROB2");
```

Dieselbe Auswirkung wie in Beispiel 1 oben.

Syntax

```
CJointT '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs jointtarget.

Weitere Informationen

Information über	Siehe
Definition einer Achse	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Lesen des aktuellen Motorwinkels	*ReadMotor - Liest die aktuellen Motorwinkel. auf Seite 1462*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.42 ClkRead - Ablesen einer Uhr, die zur Zeitnahme verwendet wird

Verwendung

`ClkRead` wird verwendet, um eine Uhr abzulesen, die als Stoppuhr für eine Zeitnahme dient.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `ClkRead`.

Beispiel 1:

```
reg1:=ClkRead(clock1);
```

Die Uhr `clock1` wird abgelesen und die Zeit wird in Sekunden in der Variablen `reg1` gespeichert.

Beispiel 2:

```
reg1:=ClkRead(clock1 \HighRes);
```

Die Uhr `clock1` wird abgelesen und die Zeit in Sekunden wird mit hoher Auflösung in der Variablen `reg1` gespeichert.

Rückgabewert

Datentyp: `num`

Die Zeit in Sekunden, die in der Uhr gespeichert ist. Die Auflösung ist normalerweise 0,001 Sekunden. Bei Verwendung des Schalters `HighRes` kann eine Auflösung von 0,000001 Sekunden erreicht werden.

Argument

```
ClkRead (Clock \HighRes)
```

`Clock`

Datentyp: `clock`

Der Name der abzulesenden Uhr.

`[ \HighRes ]`

High Resolution

Datentyp: `switch`

Legt fest, dass die Zeit mit einer höheren Auflösung abgelesen werden soll. Bei Verwendung dieses Schalters kann die Zeit mit einer Auflösung von 0,000001 Sekunden abgelesen werden.

Wegen der Präzision des Datentyps num kann die Auflösung in Mikrosekunden nur erhalten werden, solange der Ablesewert kleiner als 1 Sekunde ist.

Programmabarbeitung

Eine Uhr kann in laufendem oder gestopptem Zustand abgelesen werden.

Eine abgelesene Uhr kann erneut abgelesen, gestartet, gestoppt oder zurückgestellt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_OVERFLOW	Die Uhr läuft 4.294.967 Sekunden lang (49 Tage, 17 Stunden, 2 Minuten und 47 Sekunden), dann entsteht ein Überlauf.

Bei Verwendung des Schalters HighRes kann der Fehler ERR_OVERFLOW nicht auftreten, die Uhr wird jedoch nach etwa 49.700 Tagen ihren maximalen numerischen Wert erreichen (Wrap-Around).

Syntax

```
ClkRead '('
  [ Clock ':=' ] < variable (VAR) of clock >
  [ '\' HighRes] ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Instruktionen für Uhren	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Beispiele	*ClkStart - Starten einer Uhr, die zur Zeitnahme verwendet wird auf Seite 149*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.43 CorrRead - Lesen der aktuellen Gesamt-Offsets

Verwendung

`CorrRead` wird verwendet, um die gesamten Korrekturen zu lesen, die von allen verbundenen Korrekturgeneratoren geliefert werden.

`CorrRead` kann verwendet werden zum:

- Ermitteln, wie weit die aktuelle Bahn von der originalen Bahn abweicht.
- Ergreifen von Maßnahmen, um die Differenz zu verringern.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CorrRead`.

Siehe auch *Weitere Beispiele auf Seite 1252*.

Beispiel 1:

```
VAR pos offset;
...
offset := CorrRead();
```

Die aktuellen Offsets, die von allen verbundenen Korrekturgeneratoren geliefert werden, sind in der Variablen "offset" verfügbar.

Rückgabewert

Datentyp: `pos`

Der gesamte, absolute Offset, der sich bisher aus allen angeschlossenen Korrekturgeneratoren ergibt.

Weitere Beispiele

Weitere Beispiele für die Funktion `CorrRead` finden Sie unter der Instruktion `CorrCon`.

Syntax

```
CorrRead '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pos`.

Weitere Informationen

Information über	Siehe
Aufbauen einer Verbindung zu einem Korrekturgenerator	*CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*
Trennen der Verbindung zu einem Korrekturgenerator	*CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator auf Seite 182*
Schreiben an einen Korrekturgenerator	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Entfernen aller Korrekturgeneratoren	*CorrClear - Entfernen aller Korrekturgeneratoren auf Seite 176*
Korrekturdeskriptor	*corrdescr – Deskriptor des Korrekturgenerators auf Seite 1685*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.44 Cos - Berechnen des Cosinus

Verwendung

`Cos` (*Cosine*) wird zur Berechnung des Cosinuswertes aus einem Winkel für die Datentypen `num` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Cos`.

Beispiel 1:

```
VAR num angle;
VAR num value;
...
...
value := Cos(angle);
```

`value` ruft den Cosinuswert von `angle` ab.

Rückgabewert

Datentyp: `num`

Der Cosinuswert, Bereich = [-1, 1].

Argumente

```
Cos (Angle)
```

`Angle`

Datentyp: `num`

Der Wert des Winkels in Grad.

Syntax

```
Cos '('
  [Angle ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.45 CosDnum - Berechnen des Cosinus

Verwendung

`CosDnum` (*Cosine dnum*) wird zur Berechnung des Cosinuswertes aus einem Winkel für die Datentypen `dnum` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CosDnum`.

Beispiel 1:

```
VAR dnum angle;
VAR dnum value;
...
...
value := CosDnum(angle);
```

`value` ruft den Cosinuswert von `angle` ab.

Rückgabewert

Datentyp: `dnum`

Der Cosinuswert, Bereich = [-1, 1].

Argumente

```
CosDnum (Angle)
```

`Angle`

Datentyp: `dnum`

Der Wert des Winkels in Grad.

Syntax

```
CosDnum '('
  [Angle ':='] <expression (IN) of dnum> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.46 CPos - Lesen der aktuellen Positionsdaten (pos)

Verwendung

`CPos` *(Current Position)* wird verwendet, um die aktuelle Position des Roboters zu lesen.

Diese Funktion gibt den x-, y- und z-Wert des Roboter-TCPs als Daten des Typs pos zurück. Wenn die vollständige Roboterposition (`robtarget`) gelesen werden soll, verwenden Sie stattdessen die Funktion `CRobT`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CPos`.

Siehe auch *Weitere Beispiele auf Seite 1256*.

Beispiel 1:

```
VAR pos pos1;

MoveL *, v500, fine \Inpos := inpos50, tool1;
pos1 := CPos(\Tool:=tool1 \WObj:=wobj0);
```

Die aktuelle Position des Roboter-TCP wird in der Variablen `pos1` gespeichert. Das Werkzeug `tool1` und das Werkobjekt `wobj0` werden zur Berechnung der Position verwendet.

Beachten Sie, dass die Position erst abgelesen und berechnet wird, wenn sich der Roboter im Stillstand befindet. Dies erreichen Sie, indem Sie in der vorangehenden Bewegungsinstruktion den Stopppunkt `fine` in der Positionsgenauigkeit `inpos50` verwenden.

Rückgabewert

Datentyp: `pos`

Die aktuelle Position des Roboters mit x, y und z im äußersten Koordinatensystem, wobei das angegebene Werkzeug, das Werkobjekt und das aktive ProgDisp-Koordinatensystem berücksichtigt werden.

Argumente

```
CPos([\Tool] [\WObj])
```

`[\Tool]`

Datentyp: `tooldata`

Werkzeug zur Berechnung der aktuellen Roboterposition.

Wenn dieses Argument weggelassen wird, wird das aktuelle aktive Werkzeug verwendet.

`[\WObj]`

Work Object

Datentyp: `wobjdata`

Das Werkobjekt (Koordinatensystem), auf das sich die von der Funktion zurückgegebene aktuelle Roboterposition bezieht.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn dieses Argument weggelassen wird, wird das aktuelle aktive Werkobjekt verwendet.

> **WARNUNG**
>
> Es empfiehlt sich, bei der Programmierung immer die Argumente `\Tool` und `\WObj` anzugeben. Die Funktion gibt dann stets die gewünschte Position zurück, selbst wenn ein anderes Werkzeug oder Werkobjekt aktiviert ist.

Programmabarbeitung

Die zurückgegebenen Koordinaten stellen die TCP-Position im ProgDisp-Koordinatensystem dar.

Weitere Beispiele

Weitere Beispiele für die Funktion `CPos` sind unten aufgeführt.

```
VAR pos pos2;
VAR pos pos3;
VAR pos pos4;

pos2 := CPos(\Tool:=grip3 \WObj:=fixture);
...
pos3 := CPos(\Tool:=grip3 \WObj:=fixture);
pos4 := pos3-pos2;
```

Die x-, y- und z-Position des Roboters wird mithilfe der Funktion `CPos` an zwei Stellen im Programm erfasst. Das Werkzeug `grip3` und das Werkobjekt `fixture` werden zur Berechnung der Position verwendet. Die zwischen diesen Positionen zurückgelegten x-, y- und z-Abstände werden dann berechnet und in der Variablen `pos4` gespeichert.

Syntax

```
CPos '('
  ['\' Tool ':=' <persistent (PERS) of tooldata>]
  ['\' WObj ':=' <persistent (PERS) of wobjdata>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pos`.

Weitere Informationen

Information über	Siehe
Definition der Position	*pos – Positionen (nur X, Y und Z) auf Seite 1757*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
ProgDisp-Koordinatensystem	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Lesen des aktuellen `robtarget`	*CRobT - Lesen der aktuellen Positionsdaten (Roboterziel) auf Seite 1257*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.47 CRobT - Lesen der aktuellen Positionsdaten (Roboterziel)

Verwendung

`CRobT` *(Current Robot Target)* wird verwendet, um die aktuelle Position der Roboterachse und der externen Achsen zu lesen.

Diese Funktion gibt einen Wert des Typs `robtarget` mit Position (x, y, z), Orientierung (q1 ... q4), Roboterachsenkonfiguration und Position der externen Achsen zurück. Wenn nur x-, y- und z-Wert des Roboter-TCP (`pos`) gelesen werden sollen, verwenden Sie stattdessen die Funktion `CPos`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CRobT`.

Siehe auch *Weitere Beispiele auf Seite 1259*.

Beispiel 1:

```
VAR robtarget p1;
MoveL *, v500, fine \Inpos := inpos50, tool1;
p1 := CRobT(\Tool:=tool1 \WObj:=wobj0);
```

Die aktuelle Position der Roboterachse und der externen Achsen werden in `p1` gespeichert. Das Werkzeug `tool1` und das Werkobjekt `wobj0` werden zur Berechnung der Position verwendet.

Beachten Sie, dass die Position erst abgelesen und berechnet wird, wenn sich der Roboter im Stillstand befindet. Dies erreichen Sie, indem Sie in der vorangehenden Bewegungsinstruktion den Stopppunkt `fine` in der Positionsgenauigkeit `inpos50` verwenden.

Rückgabewert

Datentyp: `robtarget`

Die aktuelle Position des Roboters und der externen Achsen wird im äußersten Koordinatensystem angegeben, unter Berücksichtigung des angegebenen Werkzeugs, Werkobjekts und des aktiven `ProgDisp/ExtOffs`-Koordinatensystems.

Argumente

```
CRobT ([\TaskRef]|[\TaskName] [\Tool] [\WObj])
```

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, aus der das `robtarget` gelesen werden soll.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“; für den Task `T_ROB1` lautet die Variablenidentität z. B. `T_ROB1Id`.

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, aus der das `robtarget` gelesen werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.47 CRobT - Lesen der aktuellen Positionsdaten (Roboterziel)
RobotWare - OS
Fortsetzung

Wenn keines der Argumente \TaskRef oder \TaskName angegeben ist, wird die aktuelle Task verwendet.

[\Tool]

Datentyp: tooldata

Die persistente Variable für das Werkzeug, die zur Berechnung der aktuellen Roboterposition verwendet wird.

Wenn dieses Argument weggelassen wird, wird das aktuelle aktive Werkzeug verwendet.

[\WObj]

Work Object

Datentyp: wobjdata

Die persistente Variable für das Werkobjekt (Koordinatensystem), auf das sich die von der Funktion zurückgegebene aktuelle Roboterposition bezieht.

Wenn dieses Argument weggelassen wird, wird das aktuelle aktive Werkobjekt verwendet.

WARNUNG

Es empfiehlt sich, bei der Programmierung immer die Argumente \Tool und \WObj anzugeben. Die Funktion gibt dann stets die gewünschte Position zurück, selbst wenn ein anderes Werkzeug oder Werkobjekt aktiviert ist.

Programmabarbeitung

Die zurückgegebenen Koordinaten stellen die TCP-Position im ProgDisp-Koordinatensystem dar. Externe Achsen werden im ExtOffs-Koordinatensystem angegeben.

Wenn das Argument \TaskRef oder \TaskName verwendet wird, jedoch nicht die Argumente Tool und WObj, werden das aktuelle Werkzeug und Werkobjekt in der angegebenen Task verwendet.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.
ERR_NOT_MOVETASK	Argument \TaskRef oder \TaskName gibt eine Nicht-Bewegungstask an.

Es wird jedoch kein Fehler generiert, wenn die Argumente \TaskRef oder \TaskName die Nicht-Bewegungstask angeben, die die Funktion CRobT ausführt (Verweis auf die eigene Nicht-Bewegungstask). Die Position wird dann aus der verbundenen Bewegungstask abgerufen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Weitere Beispiele für die Funktion `CRobT` sind unten aufgeführt.

Beispiel 1:

```
VAR robtarget p2;
p2 := ORobT( CRobT(\Tool:=grip3 \WObj:=fixture) );
```

Die aktuelle Position der Roboterachse und der externen Achsen im Objekt-Koordinatensystem (ohne `ProgDisp` oder `ExtOffs`) wird in `p2` gespeichert. Das Werkzeug `grip3` und das Werkobjekt `fixture` werden zur Berechnung der Position verwendet.

Beispiel 2:

```
! In task T_ROB1
VAR robtarget p3;
p3 := CRobT(\TaskRef:=T_ROB2Id \Tool:=tool1 \WObj:=wobj0);
```

Die aktuellen Positionen der Roboterachse und der externen Achsen in der Task `T_ROB2` werden `p3` in der Task `T_ROB1` gespeichert. Das Werkzeug `tool1` und das Werkobjekt `wobj0` werden zur Berechnung der Position verwendet.

Beachten Sie, dass sich der Roboter in der Task `T_ROB2` möglicherweise bewegt, wenn die Position gelesen und berechnet wird. Um sicherzustellen dass der Roboter still steht, kann ein Stopppunkt `fine` in der vorherigen Bewegungsinstruktion in Task `T_ROB2` programmiert werden, und mit der Instruktion `WaitSyncTask` können die Instruktionen in der Task `T_ROB1` synchronisiert werden.

Beispiel 3

```
! In task T_ROB1
VAR robtarget p4;
p4 := CRobT(\TaskName:="T_ROB2");
```

Die aktuellen Positionen der Roboterachse und der externen Achsen in der Task `T_ROB2` werden `p4` in der Task `T_ROB1` gespeichert. Das aktuelle Werkzeug und Werkobjekt in der Task `T_ROB2` werden zur Berechnung der Position verwendet.

Syntax

```
CRobT '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>]
  ['\'Tool ':=' <persistent (PERS) of tooldata>]
  ['\' WObj ':=' <persistent (PERS) of wobjdata>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `robtarget`.

Weitere Informationen

Information über	Siehe
Definition der Position	*robtarget - Positionsdaten auf Seite 1778*
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.47 CRobT - Lesen der aktuellen Positionsdaten (Roboterziel)

RobotWare - OS

Fortsetzung

Information über	Siehe
ProgDisp-Koordinatensystem	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
ExtOffs-Koordinatensystem	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*
Ablesen der aktuellen `pos` (nur x, y, z)	*CPos - Lesen der aktuellen Positionsdaten (pos) auf Seite 1255*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.48 CrossProd - Kreuzprodukte von zwei Positionsvektoren berechnen.

Verwendung

`CrossProd` (*Cross Product*) wird verwendet, um das Kreuzprodukt (oder Vektorprodukt) von zwei `pos` Vektoren zu berechnen.

Das Kreuzprodukt von zwei Vektoren **A** und **B** ist ein Vektor, der senkrecht zu beiden Argumentvektoren ist. Die Länge des Ergebnisvektors ist gleich den Produkten der Längen von **A** und **B** und dem Sinus des Winkels zwischen ihnen θ_{AB}.

$$|A \times B| = |A||B| \sin \theta_{AB}$$

Hinweis

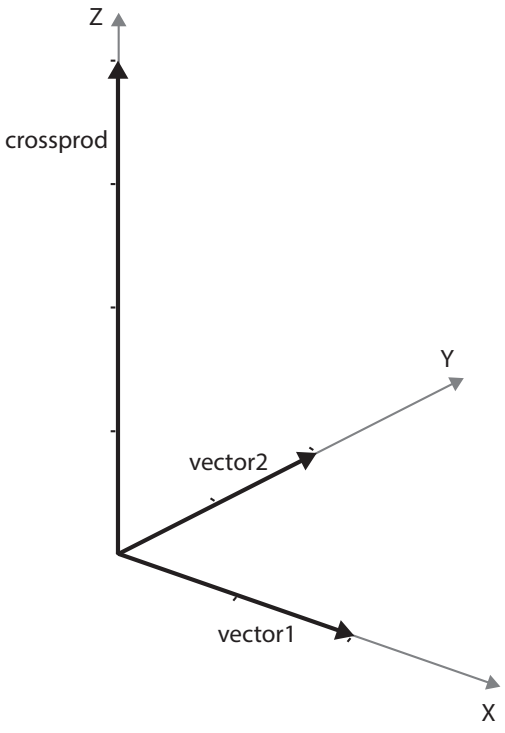
- Die Größe des Kreuzprodukts entspricht der Fläche eines Parallelogramms mit den Vektoren als Seiten.
- Das Kreuzprodukt zweier paralleler Vektoren ist null.
- **A x B = - B x A**

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `CrossProd` mit senkrechten Vektoren.

Andere Beispiele siehe *Weitere Beispiele auf Seite 1262*.

Beispiel 1:



xx1700001570

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.48 CrossProd - Kreuzprodukte von zwei Positionsvektoren berechnen.

RobotWare - OS

Fortsetzung

```
VAR pos crossprod_1;
VAR pos vector1;
VAR pos vector2;
...
...
vector1 := [2,0,0];
vector2 := [0,2,0];
crossprod_1 := CrossProd(vector1, vector2);
```

In diesem Beispiel ist `vector1` parallel zur x-Achse, `vector2` ist parallel zur y-Achse. Das Kreuzprodukt ist senkrecht zu beiden, d. h. parallel zur z-Achse.

Da der Winkel zwischen `vector1` und `vector2` 90° beträgt, ist die Größe des Kreuzprodukts: $2 * 2 * \sin 90^\circ = 4$

Rückgabewert

Datentyp: `pos`

Ein Vektor, der das Ergebnis des Kreuzprodukts der beiden Vektoren ist.

Argumente

```
CrossProd (Vector1 Vector2)
```

`Vector1`

Datentyp: `pos`

Der erste Vektor mit dem Datentyp `pos`.

`Vector2`

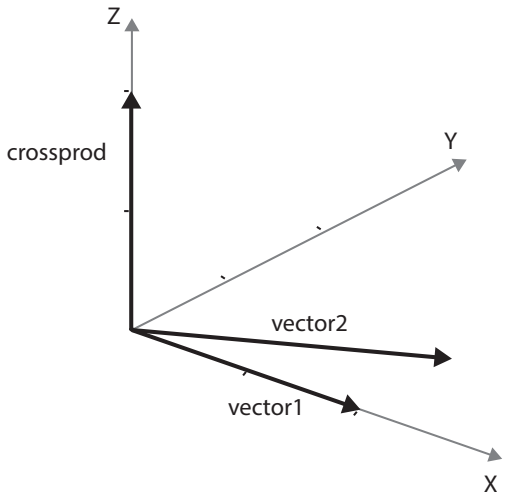
Datentyp: `pos`

Der zweite Vektor mit dem Datentyp `pos`.

Weitere Beispiele

Weitere Beispiele für die Funktion `CrossProd` sind unten aufgeführt.

Beispiel 2:



xx1700001571

```
VAR pos crossprod_1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
VAR pos vector1;
VAR pos vector2;
...
...
vector1 := [2,0,0];
vector2 := [2,1,0];
crossprod_1 := CrossProd(vector1, vector2);
```

In diesem Beispiel sind `vector1` und `vector2` parallel zur xy-Ebene. Das Kreuzprodukt ist senkrecht zu beiden, d. h. parallel zur z-Achse.

Die Größe von `vector1` ist 2. Die Größe von `vector2` ist $\sqrt{5}$. Der Winkel zwischen `vector1` und `vector2` beträgt 26,565°. Die Größe des Kreuzprodukts ist:

$$2 * \sqrt{5} * \sin 26.565^\circ = 2$$

Syntax

```
CrossProd '('
  [Vector1 ':='] <expression (IN) of pos>','
  [Vector2 ':='] <expression (IN) of pos>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pos`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.49 CSpeedOverride - Lesen der aktuellen Overridegeschwindigkeit

Verwendung

`CSpeedOverride` wird verwendet, um den Geschwindigkeitsoverride zu lesen, den der Bediener am FlexPendant festgelegt hat. Der Rückgabewert wird als Prozentsatz angezeigt, wobei die Angabe 100 % der programmierten Geschwindigkeit entspricht.

In Anwendungen mit der Instruktion `SpeedRefresh` kann diese Funktion auch verwendet werden, um den aktuellen Geschwindigkeitsoverride-Wert für diese oder die verbundene Bewegungsprogrammtask zu lesen.

Hinweis! Dies ist nicht mit dem Argument `Override` in der RAPID-Instruktion `VelSet` zu verwechseln.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CSpeedOverride`.

Beispiel 1:

```
VAR num myspeed;
myspeed := CSpeedOverride();
```

Die aktuelle Overridegeschwindigkeit wird in der Variablen `myspeed` gespeichert. Wenn der Wert zum Beispiel 100 ist, entspricht dies 100 %.

Rückgabewert

Datentyp: `num`

Die Overridegeschwindigkeit als Prozentsatz der programmierten Geschwindigkeit. Dies ist ein numerischer Wert im Bereich 0-100.

Argumente

```
CSpeedOverride ( [\CTask] )
```

`[\CTask]`

Datentyp: `switch`

Ruft den aktuellen Geschwindigkeitsoverride-Wert für diese oder die verbundene Bewegungsprogrammtask ab. Wird gemeinsam mit der Instruktion `SpeedRefresh` verwendet.

Wenn dieses Argument nicht verwendet wird, gibt die Funktion die aktuelle Geschwindigkeitsübersteuerung über das gesamte System (alle Bewegungsprogrammtasks) zurück. Dies betrifft die manuelle Geschwindigkeitsübersteuerung, die mithilfe von FlexPendant eingestellt wird.

Syntax

```
CSpeedOverride '('
  ['\' CTask ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Ändern der Overridegeschwindigkeit	*Bedienungsanleitung - IRC5 mit FlexPendant*, Abschnitt *Programming and Testing Production Running - Quickset menu, Speed*
Aktualisieren des Geschwindigkeitsoverrides über RAPID	*SpeedRefresh – Geschwindigkeitsübersteuerung für laufende Bewegung aktualisieren auf Seite 795*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.50 CTime - Lesen der aktuellen Uhrzeit als Zeichenfolge

Verwendung

`CTime` wird verwendet, um die aktuelle Systemzeit zu lesen.

Mithilfe dieser Funktion kann die aktuelle Uhrzeit am FlexPendant angezeigt oder in eine Textdatei eingefügt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CTime`.

Siehe auch *Weitere Beispiele auf Seite 1266*.

Beispiel 1:

```
VAR string time;
time := CTime();
```

Die aktuelle Zeit wird in der Variablen `time` gespeichert.

Rückgabewert

Datentyp: `string`

Das aktuelle Datum in einer Zeichenfolge.

Das Standardzeitformat ist „Stunden:Minuten:Sekunden“, z. B. 18:20:46.

Weitere Beispiele

Weitere Beispiele für die Funktion `CTime` sind unten aufgeführt.

Beispiel 1:

```
VAR string time;
time := CTime();
TPWrite "The current time is: "+time;
Write logfile, time;
```

Die aktuelle Uhrzeit wird am FlexPendant angezeigt und in eine Textdatei geschrieben.

Syntax

```
CTime '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Zeit- und Datums-Instruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - System & Time*
Einstellen der Systemuhr	*Bedienungsanleitung - IRC5 mit FlexPendant*, Abschnitt *Changing FlexPendant settings*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.51 CTool - Lesen der aktuellen Werkzeugdaten

Verwendung

`CTool` (*Current Tool*) wird verwendet, um die Daten des aktuellen Werkzeugs zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CTool`.

Beispiel 1:

```
PERS tooldata temp_tool:= [ TRUE, [ [0, 0, 0], [1, 0, 0 ,0] ],
     [0.001, [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0] ];
temp_tool := CTool();
```

Der Wert des aktuellen Werkzeugs wird in der Variablen `temp_tool` gespeichert.

Rückgabewert

Datentyp: `tooldata`

Diese Funktion gibt einen `tooldata`-Wert zurück, der das aktuelle, d. h. zuletzt in einer Bewegungsinstruktion verwendete, Werkzeug angibt.

Der zurückgegebene Wert repräsentiert die TCP-Position und Orientierung im Handgelenk-Koordinatensystem. Siehe `tooldata`.

Argumente

```
CTool ([\TaskRef]|[\TaskName])
```

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, von der die Daten des aktuellen Werkzeugs gelesen werden sollten.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“; für den Task `T_ROB1` lautet die Variablenidentität z. B. `T_ROB1Id`.

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, von der die Daten des aktuellen Werkzeugs gelesen werden sollten.

Wenn keines der Argumente `\TaskRef` oder `\TaskName` angegeben ist, wird die aktuelle Task verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.51 CTool - Lesen der aktuellen Werkzeugdaten

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.
ERR_NOT_MOVETASK	Argument \TaskRef oder \TaskName gibt eine Nicht-Bewegungstask an.

Es wird jedoch kein Fehler erzeugt, wenn die Argumente \TaskRef oder \TaskName die Task ohne Bewegung angeben, die diese Funktion ausführen CTool (Referenz zur eigenen Task ohne Bewegung). Die Werkzeugdaten werden dann von der verbundenen Bewegungstask geholt.

Syntax

```
CTool '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs tooldata.

Weitere Informationen

Information über	Siehe
Definition von Werkzeugen	*tooldata – Werkzeugdaten auf Seite 1823*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - Coordinate Systems*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.52 CWObj - Lesen der aktuellen Werkobjektdaten

Verwendung

`CWObj` (*Current Work Object*) wird verwendet, um die Daten des aktuellen Werkobjekts zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `CWObj`.

Beispiel 1:

```
PERS wobjdata temp_wobj:= [FALSE, TRUE, "", [[0,0,0], [1,0,0,0]],
     [[0,0,0], [1,0,0,0]]];
temp_wobj := CWObj();
```

Der Wert des aktuellen Werkobjekts wird in der Variablen `temp_wobj` gespeichert.

Rückgabewert

Datentyp: `wobjdata`

Diese Funktion gibt einen `wobjdata`-Wert zurück, der das aktuelle, d. h. zuletzt in einer Bewegungsinstruktion verwendete, Werkobjekt angibt.

Der zurückgegebene Wert repräsentiert die Werkobjektposition und -orientierung im Welt-Koordinatensystem. Siehe `wobjdata`.

Argumente

```
CWobj ([\TaskRef]|[\TaskName])
```

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, von der die Daten des aktuellen Werkobjekts gelesen werden sollten.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“; für den Task `T_ROB1` lautet die Variablenidentität z. B. `T_ROB1Id`.

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, von der die Daten des aktuellen Werkobjekts gelesen werden sollten.

Wenn keines der Argumente `\TaskRef` oder `\TaskName` angegeben ist, wird die aktuelle Task verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.52 CWObj - Lesen der aktuellen Werkobjektdaten
RobotWare - OS
Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.
ERR_NOT_MOVETASK	Argument \TaskRef oder \TaskName gibt eine Nicht-Bewegungstask an.

Es wird jedoch kein Fehler erzeugt, wenn die Argumente \TaskRef oder \TaskName die Task ohne Bewegung angeben, die diese Funktion ausführen CWobj (Referenz zur eigenen Task ohne Bewegung). Die Werkobjektdaten werden dann von der verbundenen Bewegungstask geholt.

Syntax

```
CWobj '('
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs wobjdata.

Weitere Informationen

Information über	Siehe
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Coordinate Systems*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.53 DecToHex - Konvertieren von Dezimal- in Hexadezimalformat

Verwendung

`DecToHex` wird verwendet, um die in einer lesbaren Zeichenfolge angegebene Zahl mit der Basis 10 in die Basis 16 zu konvertieren.

Die resultierende Zeichenfolge wird aus dem Zeichensatz [0-9,A-F,a-f] erstellt.

Diese Routine handhabt Zahlen von 0 bis 9223372036854775807dec bzw. 7FFFFFFFFFFFFFFF hex.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DecToHex`.

Beispiel 1:

```
VAR string str;

str := DecToHex("99999999");
```

Die Variable `str` erhält den Wert `"5F5E0FF"`.

Rückgabewert

Datentyp: `string`

Die Zeichenfolge, die in eine hexadezimale Darstellung der Zahl aus der Zeichenfolge des Eingangsparameters konvertiert wurde.

Argumente

```
DecToHex ( Str )
```

`Str`

String

Datentyp: `string`

Die zu konvertierende Zeichenfolge.

Syntax

```
DecToHex '('
  [ Str ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - String functions*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Basic elements*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.54 DefAccFrame - Definieren eines präzisen Koordinatensystems

Verwendung

`DefAccFrame` (*Define Accurate Frame*) wird verwendet, um ein Koordinatensystem aus drei bis zehn Originalpositionen und derselben Anzahl an verschobenen Positionen zu definieren.

Beschreibung

Ein Koordinatensystem kann definiert werden, wenn eine Gruppe von Zielen an zwei verschiedenen Positionen bekannt ist. Daher werden *dieselben physischen Positionen* verwendet, aber unterschiedlich ausgedrückt.

Betrachten Sie dies aus zwei verschiedenen Blickwinkeln:

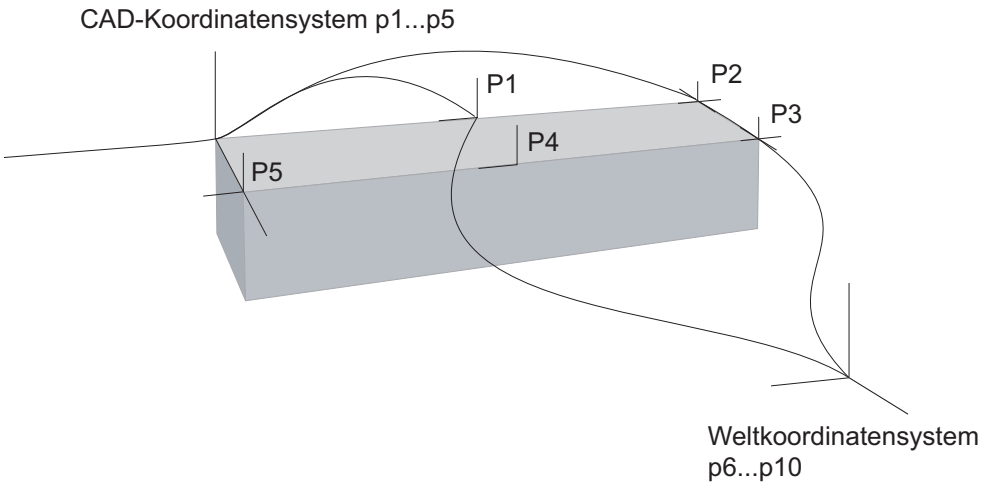
1. Es werden die gleichen physischen Positionen in Bezug auf die verschiedenen Koordinatensysteme ausgedrückt. Beispielsweise wird eine Anzahl von Positionen aus einer CAD-Zeichnung abgerufen, die Positionen werden also in einem lokalen CAD-Koordinatensystem ausgedrückt. Die gleiche Positionen werden dann Welt-Koordinatensystem des Roboters ausgedrückt. Aus diesen beiden Positionsgruppen wird das Koordinatensystem zwischen dem CAD-Koordinatensystem und dem Welt-Koordinatensystem des Roboters berechnet.
2. Eine Anzahl von Positionen bezieht sich auf ein Objekt an einer Originalposition. Nach einer Verschiebung des Objekts werden die Positionen erneut bestimmt (häufig durch Suche). Aus diesen beiden Positionsgruppen (alte Positionen, neue Positionen) wird das Verschiebungskoordinatensystem berechnet.

Drei Ziele reichen, um ein Koordinatensystem zu definieren, aber für eine höhere Genauigkeit sollten mehr Punkte verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DefAccFrame`.

Beispiel 1:



xx0500002179

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
CONST robtarget p4 := [...];
CONST robtarget p5 := [...];

VAR robtarget p6 := [...];
VAR robtarget p7 := [...];
VAR robtarget p8 := [...];
VAR robtarget p9 := [...];
VAR robtarget p10 := [...];
VAR robtarget pWCS{5};
VAR robtarget pCAD{5};

VAR pose frame1;
VAR num max_err;
VAR num mean_err;

! Add positions to robtarget arrays
pCAD{1}:=p1;
...
pCAD{5}:=p5;

pWCS{1}:=p6;
...
pWCS{5}:=p10;
frame1 := DefAccFrame (pCAD, pWCS, 5, max_err, mean_err);
```

Es wurden fünf Positionen `p1- p5` in Bezug auf ein Objekt gespeichert. Die fünf Positionen werden auch in Relation zum Welt-Koordinatensystem als `p6-p10` gespeichert. Aus diesen 10 Positionen wird das Koordinatensystem, `frame1`, zwischen de, Objekt und dem Welt-Koordinatensystem berechnet. Bei dem Koordinatensystem handelt es sich um das CAD-Koordinatensystem, das im Welt-Koordinatensystem ausgedrückt wird. Wenn die Eingangsfolge der Targetlists vertauscht wird, d. h. `DefAccFrame` (`pWCS, pCAD...`), dann wird das Welt-Koordinatensystem im CAD-Koordinatensystem ausgedrückt.

Rückgabewert

Datentyp: `pose`

Das berechnete `TargetListOne`-Koordinatensystem ausgedrückt im `TargetListTwo`-Koordinatensystem.

Argumente

```
DefAccFrame (TargetListOne TargetListTwo TargetsInList
     MaxErrMeanErr)
```

`TargetListOne`

Datentyp: `robtarget`

Datenfeld mit Roboterzielen, das die Positionen enthält, die in Koordinatensystem 1 definiert wurden. Die Mindestanzahl der Roboterziele ist 3, die Höchstanzahl 10.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.54 DefAccFrame - Definieren eines präzisen Koordinatensystems

RobotWare - OS

Fortsetzung

`TargetListTwo`

Datentyp: `robtarget`

Datenfeld mit Roboterzielen, das die Positionen enthält, die in Koordinatensystem 2 definiert wurden. Die Mindestanzahl der Roboterziele ist 3, die Höchstanzahl 10.

`TargetsInList`

Datentyp: `num`

Anzahl der Roboterziele in einem Datenfeld.

`MaxErr`

Datentyp: `num`

Geschätzter maximaler Fehler in mm.

`MeanErr`

Datentyp: `num`

Geschätzter mittlerer Fehler in mm.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FRAME`	Die Positionen weisen nicht die erforderliche Relation auf oder sie sind nicht genau genug angegeben.

Syntax

```
DefAccFrame '('
  [TargetListOne ':='] <array {*} (IN) of robtarget> ','
  [TargetListTwo ':='] <array {*} (IN) of robtarget> ','
  [TargetsInList ':='] <expression (IN) of num> ','
  [MaxErr ':='] <variable (VAR) of num> ','
  [MeanErr ':='] <variable (VAR) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Weitere Informationen

Information über	Siehe
Berechnen eines Koordinatensystems aus drei Positionen	*DefFrame - Definieren eines Koordinatensystems auf Seite 1278*
Berechnen eines Koordinatensystems aus 6 Positionen	*DefDFrame - Definieren eines Verschiebungs-Koordinatensystems auf Seite 1275*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.55 DefDFrame - Definieren eines Verschiebungs-Koordinatensystems

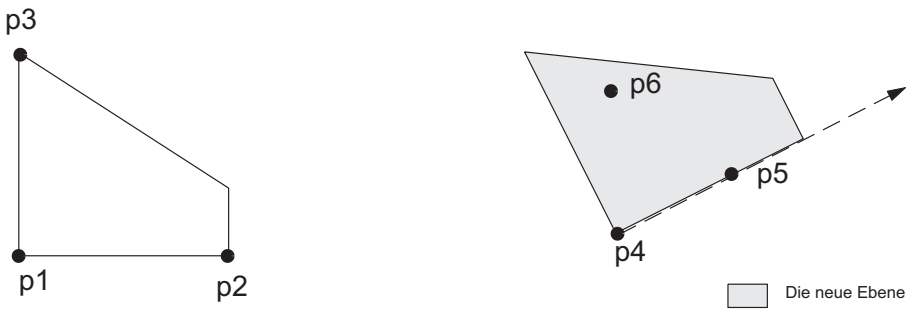
Verwendung

`DefDFrame`(*Define Displacement Frame*) wird verwendet, um ein Verschiebungs-Koordinatensystem aus drei Originalpositionen und drei verschobenen Positionen zu definieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DefDFrame`.

Beispiel 1:



xx0500002177

```
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
VAR robtarget p4;
VAR robtarget p5;
VAR robtarget p6;
VAR pose frame1;
...
!Search for the new positions
SearchL sen1, p4, *, v50, tool1;
...
SearchL sen1, p5, *, v50, tool1;
...
SearchL sen1, p6, *, v50, tool1;
frame1 := DefDframe (p1, p2, p3, p4, p5, p6);
...
!Activation of the displacement defined by frame1
PDispSet frame1;
```

Die drei Positionen `p1-p3`, die sich auf ein Objekt an einer Originalposition beziehen, wurden gespeichert. Nach einer Verschiebung des Objekts werden drei neue Positionen gesucht und als `p4-p6` gespeichert. Das Verschiebungs-Koordinatensystem wird aus diesen sechs Positionen berechnet. Dann werden mithilfe des berechneten Koordinatensystems alle im Programm gespeicherten Positionen verschoben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.55 DefDFrame - Definieren eines Verschiebungs-Koordinatensystems

RobotWare - OS

Fortsetzung

Rückgabewert

Datentyp: pose

Das Verschiebungs-Koordinatensystem.

Argumente

```
DefDFrame (OldP1 OldP2 OldP3 NewP1 NewP2 NewP3)
```

OldP1

Datentyp: robtarget

Die erste Originalposition.

OldP2

Datentyp: robtarget

Die zweite Originalposition.

OldP3

Datentyp: robtarget

Die dritte Originalposition.

NewP1

Datentyp: robtarget

Die erste verschobene Position. Die Differenz zwischen OldP1 **und** NewP1 **definiert die Translation des Koordinatensystems und muss mit großer Präzision gemessen und bestimmt werden.**

NewP2

Datentyp: robtarget

Die zweite verschobene Position. Die Linie NewP1 ... NewP2 **definiert die Rotation der alten Linie** OldP1 ... OldP2**.**

NewP3

Datentyp: robtarget

Die dritte verschobene Position. Diese Position definiert die Drehung der Ebene, z. B. sollte sie auf der neuen Ebene von NewP1**,** NewP2 **und** NewP3 **angeordnet werden.**

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO **wird gesetzt auf:**

Name	Fehlerursache
ERR_FRAME	Das Koordinatensystem kann aufgrund mangelnder Genauigkeit der Positionen nicht berechnet werden.

Syntax

```
DefDFrame '('
  [OldP1 ':='] <expression (IN) of robtarget> ','
  [OldP2 ':='] <expression (IN) of robtarget> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[OldP3 ':='] <expression (IN) of robtarget> ','
[NewP1 ':='] <expression (IN) of robtarget> ','
[NewP2 ':='] <expression (IN) of robtarget> ','
[NewP3 ':='] <expression (IN) of robtarget> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Weitere Informationen

Information über	Siehe
Aktivieren eines Verschiebungs-Koordinatensystems	*PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem auf Seite 542*
Manuelle Definition des Verschiebungs-Koordinatensystems	*Bedienungsanleitung - IRC5 mit FlexPendant*, Abschnitt *Calibrating*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.56 DefFrame - Definieren eines Koordinatensystems

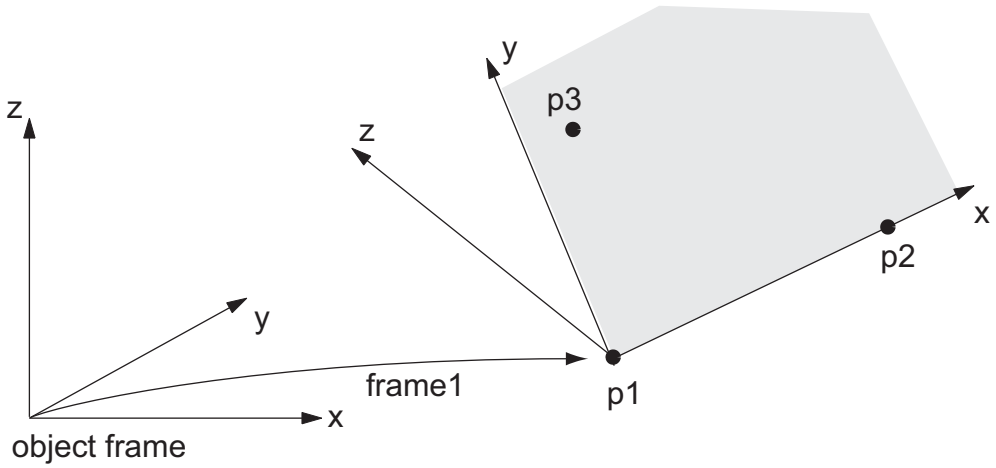
Verwendung

`DefFrame` (*Define Frame*) wird verwendet, um ein Koordinatensystem aus drei Positionen zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DefFrame`.

Beispiel 1:



xx0500002181

Die drei Positionen `p1- p3`, die sich auf das Objekt-Koordinatensystem beziehen, werden zur Definition des neuen Koordinatensystems `frame1` verwendet. Die erste Position, `p1`, definiert den Ursprung des neuen Koordinatensystems. Die zweite Position, `p2`, definiert die Richtung der x-Achse. Die dritte Position, `p3`, definiert die Position der xy-Ebene. Das definierte Koordinatensystem `frame1` kann als Verschiebungs-Koordinatensystem benutzt werden, wie in nachstehendem Beispiel illustriert:

```
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];
VAR pose frame1;
...
...
frame1 := DefFrame (p1, p2, p3);
...
...
!Activation of the displacement defined by frame1
PDispSet frame1;
```

Rückgabewert

Datentyp: `pose`

Das berechnete Koordinatensystem.

Die Berechnung bezieht sich auf das aktive Objekt-Koordinatensystem.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
DefFrame (NewP1 NewP2 NewP3 [\Origin])
```

`NewP1`

Datentyp: `robtarget`

Die erste Position, die den Ursprung des neuen Koordinatensystems definieren wird.

`NewP2`

Datentyp: `robtarget`

Die zweite Position, die die Richtung der x-Achse des neuen Koordinatensystems definieren wird.

`NewP3`

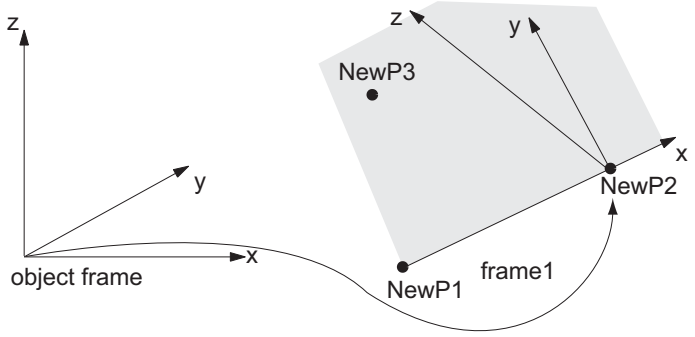
Datentyp: `robtarget`

Die dritte Position, die die xy-Ebene des neuen Koordinatensystems definieren wird. Die Position von Punkt 3 wird sich auf der positiven y-Seite befinden (siehe obige Abbildung).

`[\Origin]`

Datentyp: `num`

Optionales Argument, das definiert, wie der Ursprung des neuen Koordinatensystems positioniert wird. `Origin` = 1 bedeutet, dass sich der Ursprung in `NewP1` befindet, d. h. am gleichen Ursprung wie bei Weglassen dieses Arguments. `Origin` = 2 bedeutet, dass sich der Ursprung in `NewP2` befindet. (siehe nachfolgende Tabelle).



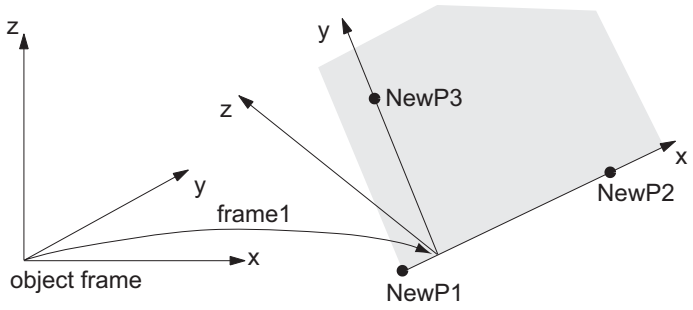
xx0500002178

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.56 DefFrame - Definieren eines Koordinatensystems
RobotWare - OS
Fortsetzung

`Origin` = 3 bedeutet, dass sich der Ursprung auf der Linie befindet, die so durch `NewP1` und `NewP2` verläuft, dass sich `NewP3` auf der y-Achse befindet. (siehe nachfolgende Tabelle).



xx0500002180

Andere Werte oder das Weglassen von `Origin` platzieren den Ursprung in `NewP1`.

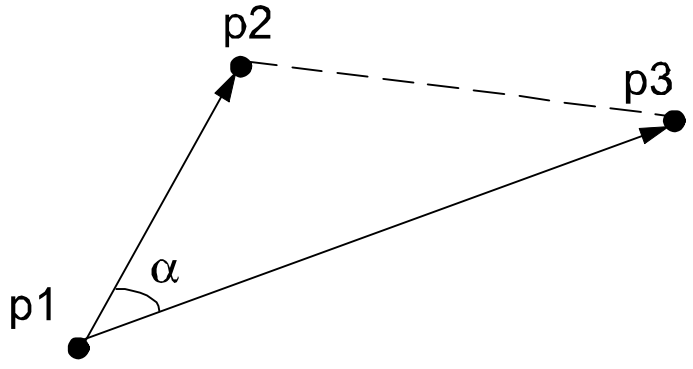
Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
ERR_FRAME	Das Koordinatensystem kann wegen der unten aufgeführten Einschränkungen nicht berechnet werden.

Einschränkungen

Die drei Positionen `p1 - p3`, die das Koordinatensystem definieren, müssen ein wohlgeformtes Dreieck definieren. Die drei Seiten des Dreiecks sollten möglichst gleich lang sein.



xx0500002182

Das Dreieck wird nicht als wohlgeformt betrachtet, wenn der Winkel á zu klein ist. Der Winkel á ist zu klein, wenn:

$|\cos \alpha| < 1 - 10^{-4}$

Das Dreieck `p1`, `p2`, `p3` darf nicht zu klein sein, d. h. die Positionen dürfen nicht zu eng beieinander liegen. Die Abstände zwischen den Positionen `p1 - p2` und `p1 - p3` dürfen nicht weniger als 0,1 mm sein.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
DefFrame '('
  [NewP1 ':='] <expression (IN) of robtarget> ','
  [NewP2 ':='] <expression (IN) of robtarget> ','
  [NewP3 ':='] <expression (IN) of robtarget>
  ['\' Origin ':=' <expression (IN) of num >] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*
Aktivieren eines Verschiebungs-Koordinatensystems	*PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem auf Seite 542*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.57 Dim - Abrufen der Dimension eines Datenfelds

Verwendung

`Dim` (*Dimension*) wird verwendet, um die Anzahl der Elemente in einem Datenfeld abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Dim`.

Siehe auch *Weitere Beispiele auf Seite 1282*.

Beispiel 1:

```
PROC arrmul(VAR num array{*}, num factor)
  FOR index FROM 1 TO Dim(array, 1) DO
    array{index} := array{index} * factor;
  ENDFOR
ENDPROC
```

Alle Elemente in einem num-Datenfeld werden mit einem Faktor multipliziert. Diese Prozedur kann ein beliebiges eindimensionales Datenfeld des Typs `num` als Eingabe haben.

Rückgabewert

Datentyp: `num`

Die Anzahl der Datenfeldelemente der angegebenen Dimension.

Argumente

```
Dim (ArrPar DimNo)
```

`ArrPar`

Array Parameter

Datentyp: `anytype`

Name des Datenfelds.

`DimNo`

Dimension Number

Datentyp: `num`

Die gewünschte Datenfelddimension.

1 = erste Dimension

2 = zweite Dimension

3 = dritte Dimension

Weitere Beispiele

Weitere Beispiele für die Verwendung der Funktion `Dim` sind unten aufgeführt.

Beispiel 1:

```
PROC add_matrix(VAR num array1{*,*,*}, num array2{*,*,*})
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  IF Dim(array1,1) <> Dim(array2,1) OR Dim(array1,2) <>
       Dim(array2,2) OR Dim(array1,3) <> Dim(array2,3) THEN
    TPWrite "The size of the matrices are not the same";
    Stop;
  ELSE
    FOR i1 FROM 1 TO Dim(array1, 1) DO
      FOR i2 FROM 1 TO Dim(array1, 2) DO
        FOR i3 FROM 1 TO Dim(array1, 3) DO
          array1{i1,i2,i3} := array1{i1,i2,i3} + array2{i1,i2,i3};
        ENDFOR
      ENDFOR
    ENDFOR
  ENDIF
  RETURN;

ENDPROC
```

Zwei Matrizen werden hinzugefügt. Wenn sich die Größe der Matrizen unterscheidet, stoppt das Programm und eine Fehlermeldung wird angezeigt.

Diese Prozedur kann beliebige dreidimensionale Datenfelder des Typs `num` als Eingabe haben.

Syntax

```
Dim '('
  [ArrPar ':='] <reference (REF) of anytype> ','
  [DimNo ':='] <expression (IN) of num> ')'
```

Ein REF-Parameter verlangt, dass das entsprechende Argument eine Konstante, eine Variable oder eine vollständige Speichernde ist. Das Argument kann auch ein IN-Parameter, ein VAR-Parameter oder ein vollständiger PERS-Parameter sein.

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Datenfeld-Parameter	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Routines*
Datenfeld-Deklaration	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.58 DInput - Lesen des Wertes eines digitalen Eingangssignals

Verwendung

`DInput` wird verwendet, um den aktuellen Wert eines digitalen Eingangssignals zu lesen.

> **Hinweis**
>
> Beachten Sie, dass die Funktion `DInput` eine alte Funktion ist, die nicht länger verwendet werden darf. Beachten Sie die Beispiele für eine alternative und empfohlene Art der Programmierung.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DInput`.

Siehe auch *Weitere Beispiele auf Seite 1284*.

Beispiel 1:

```
IF DInput(di2) = 1 THEN ...
...
IF di2 = 1 THEN ...
```

Wenn der aktuelle Wert des Signals `di2` gleich 1 ist, dann...

Rückgabewert

Datentyp: `num`

Der aktuelle Signalwert (0 oder 1).

Argumente

```
DInput (Signal)
```

`Signal`

Datentyp: `signaldi`

Der Name des digitalen Eingangs, der gelesen werden soll.

Programmabarbeitung

Der gelesene Wert hängt von der Konfiguration des Signals ab. Wenn das Signal in den Systemparametern invertiert ist, ist der zurückgegebene Wert dem Wert des physischen Kanals entgegengesetzt.

Weitere Beispiele

Weitere Beispiele für die Verwendung der Funktion `DInput` sind unten aufgeführt.

Beispiel 1:

```
weld_flag := DInput(weld);
...
weld_flag := weld;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Variable `weld_flag` wird auf denselben Wert gesetzt wie der aktuelle Wert des Signals `weld`.

Hinweis

Beachten Sie, dass in diesem Fall `weld_flag` den aktuellen Wert des Signals darstellt. Daher können Sie, wenn `weld_flag` später in dem Programm verwendet wird, nicht sicher sein, dass der aktuelle Wert des Signals dargestellt wird.

Syntax

```
DInput '('
  [Signal ':='] <variable (VAR) of signaladi>')'
```

Eine Funktion mit einem Rückgabewert des Typs `dionum`.

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.59 Distance – Abstand zwischen zwei Punkten

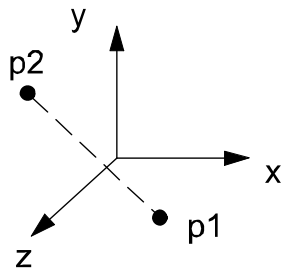
Verwendung

`Distance` wird verwendet, um den Abstand zwischen zwei Punkten im Raum zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Distance`.

Beispiel 1:



xx0500002321

```
VAR num dist;
CONST pos p1 := [4,0,4];
CONST pos p2 := [-4,4,4];
...
dist := Distance(p1, p2);
```

Der Abstand im Raum zwischen den Punkten `p1` und `p2` wird berechnet und in der Variablen `dist` gespeichert.

Rückgabewert

Datentyp: `num`

Der Abstand zwischen den Punkten in mm (immer positiv).

Argumente

```
Distance (Point1 Point2)
```

`Point1`

Datentyp: `pos`

Der erste Punkt mit dem Datentyp `pos`.

`Point2`

Datentyp: `pos`

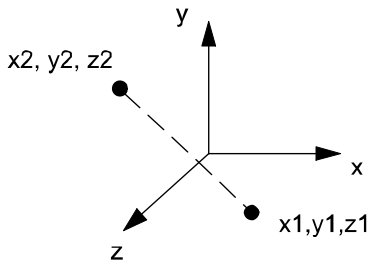
Der zweite Punkt mit dem Datentyp `pos`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Berechnung des Abstands zwischen den beiden Punkten:



xx0500002322

$$\text{Abstand} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

xx0500002323

Syntax

```
Distance '('
[Point1 ':='] <expression (IN) of pos> ','
[Point2 ':='] <expression (IN) of pos> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*
Definition der Position	*pos – Positionen (nur X, Y und Z) auf Seite 1757*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.60 DIV - berechnet eine Ganzzahldivision

Verwendung

`DIV` ist ein bedingter Ausdruck, um eine Ganzzahldivision zu berechnen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `DIV`.

Beispiel 1:

```
reg1 := 14 DIV 4;
```

Der Rückgabewert ist `3`, weil `14` durch `4` `3`-mal geteilt werden kann.

Beispiel 2:

```
VAR dnum mydnum1 := 10;
VAR dnum mydnum2 := 5;
VAR dnum mydnum3;
...
mydnum3 := mydnum1 DIV mydnum2;
```

Der Rückgabewert ist `2`, weil `10` durch `5` `2`-mal geteilt werden kann.

Rückgabewert

Datentyp: `num`, `dnum`

Gibt die Ganzzahl aus einer Division von Ganzzahlen zurück.

Syntax

```
<expression of num> DIV <expression of num>
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

```
<expression of dnum> DIV <expression of dnum>
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
num - Numerische Werte	*num - Numerische Werte auf Seite 1740*
dnum – Doppelte numerische Werte	*dnum – Doppelte numerische Werte auf Seite 1690*
MOD	*MOD - berechnet einen Ganzzahlmodulo auf Seite 1398*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.61 DnumToNum – Konvertieren von dnum in num

Verwendung

DnumToNum konvertiert gegebenenfalls ein dnum in ein num, sonst wird ein behebbarer Fehler generiert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion DnumToNum.

Beispiel 1:

```
VAR num mynum:=0;
VAR dnum mydnum:=8388607;
VAR dnum testFloat:=8388609;
VAR dnum anotherdnum:=4294967295;
! Works OK
mynum:=DnumToNum(mydnum);
! Accept floating point value
mynum:=DnumToNum(testFloat);
! Cause error recovery error
mynum:=DnumToNum(anotherdnum \Integer);
```

Der dnum-Wert 8388607 wird über die Funktion als num-Wert 8388607 zurückgegeben.

Der dnum-Wert 8388609 wird über die Funktion als num-Wert 8.38861E+06 zurückgegeben.

Der dnum-Wert 4294967295 generiert den behebbaren Fehler ERR_ARGVALERR.

Rückgabewert

Datentyp: num

Der dnum-Eingangswert kann sich im Bereich von -8388607 bis 8388608 befinden und denselben Wert zurückgeben wie ein num. Wenn der \Integer-Schalter nicht verwendet wird, kann sich der dnum-Eingangswert im Bereich von -3.40282347E+38 bis 3.40282347E+38 bewegen und der Rückgabewert kann zur Gleitkommazahl werden.

Argumente

```
DnumToNum (Value [\Integer])
```

Value

Datentyp: dnum

Der zu konvertierende numerische Wert.

[\Integer]

Datentyp: switch

Nur Ganzzahlen.

Wird der Schalter \Integer nicht verwendet, wird sogar im Falle einer Gleitkommazahl ein Downcasting vorgenommen. Wenn der Schalter verwendet

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.61 DnumToNum – Konvertieren von dnum in num

RobotWare - OS

Fortsetzung

wird, wird geprüft, ob der Wert eine Ganzzahl zwischen -8388607 und 8388608 ist. Falls nicht, wird ein behebbarer Fehler generiert.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_ARGVALERR	Der Wert liegt über 8388608 oder unter -8388607 oder ist keine Ganzzahl (wenn das optionale Argument Integer verwendet wird).
ERR_NUM_LIMIT	Der Wert liegt über 3.40282347E+38 oder unter -3.40282347E+38
ERR_INT_NOTVAL	Der Wert ist keine Ganzzahl.

Syntax

```
DnumToNum '('
  [ Value ':=' ] < expression (IN) of dnum >
  ['\' Integer ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Dnum-Datentyp	*dnum – Doppelte numerische Werte auf Seite 1690*.
num-Datentyp	*num - Numerische Werte auf Seite 1740*.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.62 DnumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge

Verwendung

`DnumToStr` (*Numeric To String*) Numerischen Wert in eine Zeichenfolge konvertieren

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `DnumToStr`.

Beispiel 1:

```
VAR string str;
str := DnumToStr(0.3852138754655357,3);
```

Die Variable `str` erhält den Wert `"0.385"`.

Beispiel 2:

```
VAR dnum val;
val:= 0.3852138754655357;
str := DnumToStr(val, 2\Exp);
```

Die Variable `str` erhält den Wert `"3.85E-01"`.

Beispiel 3

```
VAR dnum val;
val := 0.3852138754655357;
str := DnumToStr(val, 15);
```

Die Variable `str` erhält den Wert `"0.385213875465536"`.

Beispiel 4:

```
VAR dnum val;
val:=4294967295.385215;
str := DnumToStr(val, 4);
```

Die Variable `str` erhält den Wert `"4294967295.3852"`.

Rückgabewert

Datentyp: `str`

Der in eine Zeichenfolge konvertierte numerische Wert mit der angegebenen Anzahl an Dezimalstellen und mit Exponent, falls angefordert. Der numerische Wert wird bei Bedarf gerundet. Das Dezimalkomma wird unterdrückt, wenn keine Nachkommastellen vorhanden sind.

Argumente

```
DnumToStr (Val Dec [\Exp])
```

`Val`

Value

Datentyp: `dnum`

Der zu konvertierende numerische Wert.

`Dec`

Decimals

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.62 DnumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge

RobotWare - OS

Fortsetzung

Datentyp: `num`

Anzahl an Dezimalstellen. Die Anzahl an Dezimalstellen darf nicht als negativer Wert oder mit größerer Genauigkeit als die verfügbare Genauigkeit für numerische Werte angegeben werden.

Maximale Anzahl an verwendbaren Dezimalzahlen ist 15.

`[\Exp]`

Exponent

Datentyp: `switch`

Verwenden eines Exponenten im Rückgabewert.

Syntax

```
DnumToStr '('
  [ Val ':=' ] <expression (IN) of dnum>
  [ Dec ':=' ] <expression (IN) of num>
  ['\' Exp ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - String functions*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Basic elements*
Numerischen Wert in eine Zeichenfolge konvertieren	*NumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge auf Seite 1410*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.63 DotProd - Berechnen des Skalarprodukts von zwei pos-Vektoren

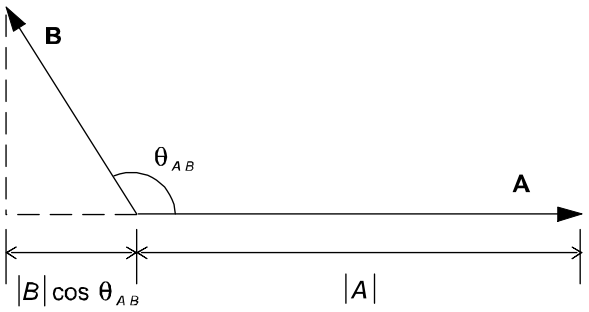
Verwendung

`DotProd` (*Dot Product*) wird verwendet, um das Skalarprodukt von zwei pos-Vektoren zu berechnen. Eine typische Anwendung ist die Berechnung der Projektion von einem Vektor auf den anderen oder die Berechnung des Winkels zwischen den beiden Vektoren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DotProd`.

Beispiel 1:



xx0500002449

Das Skalarprodukt der beiden Vektoren **A** und **B** ist ein Skalar, der dem Produkt der Längen von **A** und **B** und dem Cosinus des eingeschlossenen Winkels entspricht.

$$A \cdot B = |A||B| \cos \theta_{AB}$$

Das Skalarprodukt:

- ist kleiner oder gleich dem Produkt beider Längen.
- kann positiv oder negativ sein, abhängig davon, ob der eingeschlossene Winkel kleiner oder größer als 90 Grad ist.
- ist gleich dem Produkt der Länge eines Vektors und der Projektion des anderen Vektors auf den ersten.
- ist null, wenn die Vektoren senkrecht aufeinander stehen.

Die Vektoren werden durch den Datentyp `pos` und das Skalarprodukt durch den Datentyp `num` beschrieben:

```
VAR num dotprod;
VAR pos vector1;
VAR pos vector2;
...
...
vector1 := [1,1,1];
vector2 := [1,2,3];
dotprod := DotProd(vector1, vector2);
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.63 DotProd - Berechnen des Skalarprodukts von zwei pos-Vektoren

RobotWare - OS

Fortsetzung

Rückgabewert

Datentyp: num

Der Wert des Skalarprodukts der beiden Vektoren.

Argumente

```
DotProd (Vector1 Vector2)
```

Vector1

Datentyp: pos

Der erste Vektor mit dem Datentyp pos.

Vector2

Datentyp: pos

Der zweite Vektor mit dem Datentyp pos.

Syntax

```
DotProd '('
  [Vector1 ':='] <expression (IN) of pos>','
  [Vector2 ':='] <expression (IN) of pos>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.64 DOutput - Lesen des aktuellen Werts von einem digitalen Ausgangssignal

Verwendung

`DOutput` wird verwendet, um den aktuellen Wert eines digitalen Ausgangssignals zu lesen.

> **Hinweis**
>
> Beachten Sie, dass die Funktion `DOutput` eine alte Funktion ist, die nicht länger verwendet werden darf. Beachten Sie die Beispiele für eine alternative und empfohlene Art der Programmierung.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `DOutput`.

Siehe auch *Weitere Beispiele auf Seite 1296*.

Beispiel 1:

```
IF DOutput(do2) = 1 THEN ...
...
IF do2 = 1 THEN ...
```

Wenn der aktuelle Wert des Signals `do2` gleich `1` ist, dann `...`

Rückgabewert

Datentyp: `dionum`

Der aktuelle Signalwert (0 oder 1).

Argumente

```
DOutput (Signal)
```

`Signal`

Datentyp: `signaldo`

Der Name des zu lesenden Signals.

Programmabarbeitung

Der gelesene Wert hängt von der Konfiguration des Signals ab. Wenn das Signal in den Systemparametern invertiert ist, ist der zurückgegebene Wert dem wahren Wert des physischen Kanals entgegengesetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler können auftreten. Die Fehler können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird auf folgenden Wert gesetzt:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Wenn kein Kontakt mit dem E/A-Gerät vorhanden ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.64 DOutput - Lesen des aktuellen Werts von einem digitalen Ausgangssignal
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Weitere Beispiele

Weitere Beispiele für die Funktion `DOutput` sind unten aufgeführt.

Beispiel 1:

```
IF DOutput(auto_on) <> active THEN ...
...
IF auto_on <> active THEN ...
```

Wenn der aktuelle Wert des Systemsignals `auto_on` gleich `not active` ist, dann . . ., d. h. wenn sich der Roboter im Einrichtbetrieb befindet, dann . . .

Hinweis

Das Signal muss zuerst als Systemausgang in den Systemparametern definiert werden.

Syntax

```
DOutput '('
  [ Signal ':=' ] < variable (VAR) of signaldo > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dionum`.

Weitere Informationen

Information über	Siehe
Setzen eines digitalen Ausgangssignals	*SetDO - Ändern des Werts eines digitalen Ausgangssignals auf Seite 710*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.65 EulerZYX - Berechnen von Eulerschen Winkeln aus "orient"

Verwendung

`EulerZYX` (*Euler ZYX rotations*) wird verwendet, um die Komponente eines Eulerschen Winkels aus einer Variablen des Typs `orient` zu erhalten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `EulerZYX`.

Beispiel 1:

```
VAR num anglex;
VAR num angley;
VAR num anglez;
VAR pose object;
...
...
anglex := EulerZYX(\X, object.rot);
angley := EulerZYX(\Y, object.rot);
anglez := EulerZYX(\Z, object.rot);
```

Rückgabewert

Datentyp: `num`

Der entsprechende Eulersche Winkel in Grad, Bereich [-180, 180].

Argumente

```
EulerZYX ([\X] | [\Y] | [\Z] Rotation)
```

`[\X]`

Datentyp: `switch`

Ruft die Rotation um die X-Achse ab.

`[\Y]`

Datentyp: `switch`

Ruft die Rotation um die Y-Achse ab.

`[\Z]`

Datentyp: `switch`

Ruft die Rotation um die Z-Achse ab.

Hinweis!

Die Argumente `\X`, `\Y` und `\Z` schließen sich gegenseitig aus. Wenn keines der Argumente angegeben ist, wird ein Laufzeitfehler erzeugt.

`Rotation`

Datentyp: `orient`

Die Rotation in ihrer Quaterniondarstellung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.65 EulerZYX - Berechnen von Eulerschen Winkeln aus "orient"

RobotWare - OS

Fortsetzung

Syntax

```
EulerZYX '('
  ['\' X ','] | ['\' Y ','] | ['\' Z ',']
  [Rotation ':='] <expression (IN) of orient> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.66 EventType – Abrufen des aktuellen Ereignistyps in einer beliebigen Ereignisroutine

Verwendung

`EventType` kann in jeder Ereignisroutine verwendet werden und gibt den aktuellen ausgeführten Ereignistyp zurück.

Wenn `EventType` aus einer Programmtaskroutine aufgerufen wird, gibt `EventType` immer 0 zurück, womit EVENT_NONE angegeben wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `EventType`.

Beispiel 1:

```
TEST EventType()
CASE EVENT_NONE:
  ! Not executing any event
CASE EVENT_POWERON:
  ! Executing POWER ON event
CASE EVENT_START:
  ! Executing START event
CASE EVENT_STOP:
  ! Executing STOP event
CASE EVENT_QSTOP:
  ! Executing QSTOP event
CASE EVENT_RESTART:
  ! Executing RESTART event
CASE EVENT_RESET:
  ! Executing RESET event
CASE EVENT_STEP:
  ! Executing STEP event
ENDTEST
```

Verwenden der Funktion `EventType` in einer beliebigen Ereignisroutine, um festzustellen, welches Systemereignis (falls vorhanden) gerade abgearbeitet wird.

Rückgabewert

Datentyp: `event_type`

Der aktuelle ausgeführte Ereignistyp 1 ... 7, oder 0, wenn keine Ereignisroutine abgearbeitet wird.

Vordefinierte Daten

Folgende vordefinierte, symbolische Konstanten des Typs `event_type` können zur Überprüfung des Rückgabewerts verwendet werden.

```
CONST event_type EVENT_NONE := 0;
CONST event_type EVENT_POWERON := 1;
CONST event_type EVENT_START := 2;
CONST event_type EVENT_STOP := 3;
CONST event_type EVENT_QSTOP:= 4;
CONST event_type EVENT_RESTART := 5;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.66 EventType – Abrufen des aktuellen Ereignistyps in einer beliebigen Ereignisroutine

RobotWare - OS

Fortsetzung

```
CONST event_type EVENT_RESET := 6;
CONST event_type EVENT_STEP := 7;
```

Syntax

```
EventType '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `event_type`.

Weitere Informationen

Information über	Siehe
Ereignisroutinen im Allgemeinen	*Technisches Referenzhandbuch - Systemparameter*, Abschnitt *Controller - Event Routine*
Datentyp "event_type", vordefinierte Konstanten	*event_type – Typ der Ereignisroutine auf Seite 1704*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.67 ExecHandler - Abrufen des Typs der Abarbeitungsbehandlung

Verwendung

Mit `ExecHandler` kann ermittelt werden, ob der aktuelle `RAPID`-Code in einer `RAPID`-Programmroutinenbehandlung ausgeführt wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ExecHandler`.

Beispiel 1:

```
TEST ExecHandler()
CASE HANDLER_NONE:
  ! Not executing in any routine handler
CASE HANDLER_BWD:
  ! Executing in routine BACKWARD handler
CASE HANDLER_ERR:
  ! Executing in routine ERROR handler
CASE HANDLER_UNDO:
  ! Executing in routine UNDO handler
ENDTEST
```

Mit der Funktion `ExecHandler` wird ermittelt, ob der Code in einer Routinebehandlung ausgeführt wird.

`HANDLER_ERR` wird auch zurückgegeben, wenn der Aufruf in einer Untermethode der Fehlerbehandlung erfolgt.

Rückgabewert

Datentyp: `handler_type`

Der aktuelle ausgeführte Behandlungstyp 1 ... 3, oder 0, wenn der Code in keiner Routinenbehandlung ausgeführt wird.

Vordefinierte Daten

Folgende vordefinierte, symbolische Konstanten des Typs `handler_type` können zur Überprüfung des Rückgabewerts verwendet werden.

```
CONST handler_type HANDLER_NONE := 0;
CONST handler_type HANDLER_BWD := 1;
CONST handler_type HANDLER_ERR := 2;
CONST handler_type HANDLER_UNDO := 3;
```

Syntax

```
ExecHandler '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `handler_type`.

Weitere Informationen

Information über	Siehe
Typ der Abarbeitungsbehandlung	*handler_type – Typ des Abarbeitungshandlers auf Seite 1711*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.68 ExecLevel – Abarbeitungsebene abrufen

Verwendung

Mit `ExecLevel` kann die aktuelle Abarbeitungsebene des gegenwärtig ausgeführten RAPID-Codes ermittelt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ExecLevel`.

Beispiel 1:

```
TEST ExecLevel()
CASE LEVEL_NORMAL:
  ! Execute on base level
CASE LEVEL_TRAP:
  ! Execute in TRAP routine
CASE LEVEL_SERVICE:
  ! Execute in service, event or system input interrupt routine
ENDTEST
```

Mit der Funktion `ExecLevel` wird die aktuelle Abarbeitungsebene ermittelt.

Rückgabewert

Datentyp: `exec_level`

Die aktuelle Abarbeitungsebene 0 ... 2.

Vordefinierte Daten

Folgende vordefinierte, symbolische Konstanten des Typs `exec_level` können zur Überprüfung des Rückgabewerts verwendet werden:

```
CONST exec_level LEVEL_NORMAL := 0;
CONST exec_level LEVEL_TRAP := 1;
CONST exec_level LEVEL_SERVICE := 2;
```

Syntax

```
ExecLevel '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `exec_level`.

Weitere Informationen

Information über	Siehe
Datentyp für Abarbeitungsebene	*exec_level – Abarbeitungsebene auf Seite 1705*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.69 Exp - Berechnen des Exponentialwerts

Verwendung

`Exp` (*Exponential*) wird zur Berechnung des Exponentialwerts ex verwendet.^{x}.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Exp`.

Beispiel 1:

```
VAR num x;
VAR num value;
...
value:= Exp( x);
```

value ruft den Exponentialwert von `x` ab.

Rückgabewert

Datentyp: `num`

Der Exponentialwert e ^{x}.

Argumente

```
Exp (Exponent)
```

`Exponent`

Datentyp: `num`

Der Wert des Exponentenarguments.

Syntax

```
Exp '('
  [Exponent ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.70 FileSize - Abrufen der Größe einer Datei

Verwendung

`FileSize` wird verwendet, um die Größe der angegebenen Datei abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `FileSize`.

Siehe auch *Weitere Beispiele auf Seite 1305*.

Beispiel 1:

```
PROC listfile(string filename)
  VAR num size;
  size := FileSize(filename);
  TPWrite filename+" size: "+NumToStr(size,0)+" Bytes";
ENDPROC
```

Diese Prozedur gibt den Namen der angegebenen Datei zusammen mit ihrer Größe aus.

Rückgabewert

Datentyp: `num`

Die Größe in Byte.

Argumente

```
FileSize (Path)
```

`Path`

Datentyp: `string`

Der angegebene Dateiname mit dem vollständigen oder relativen Pfad.

Programmabarbeitung

Diese Funktion gibt einen numerischen Wert zurück, der die Größe der angegebenen Datei in Byte darstellt.

Zu Verzeichnissen sind dieselben Informationen verfügbar.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Die Datei ist nicht vorhanden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Ein grundlegendes Beispiel für die Funktion ist unten aufgeführt.

Beispiel 1:

Dieses Beispiel listet alle Dateien, die größer als `1` KB sind und sich unter der Verzeichnisstruktur "`HOME`:" einschließlich aller Unterverzeichnisse befinden, auf.

```
PROC searchdir(string dirname, string actionproc)
  VAR dir directory;
  VAR string filename;
  IF IsFile(dirname \Directory) THEN
    OpenDir directory, dirname;
    WHILE ReadDir(directory, filename) DO
        ! .. and . is the parent and resp. this directory
        IF filename <> ".." AND filename <> "." THEN
          searchdir dirname+"/"+filename, actionproc;
        ENDIF
    ENDWHILE
    CloseDir directory;
  ELSE
    %actionproc% dirname;
  ENDIF
ERROR
  RAISE;
ENDPROC

PROC listfile(string filename)
  IF FileSize(filename) > 1024 THEN
      TPWrite filename;
  ENDIF
ENDPROC

PROC main()
  ! Execute the listfile routine for all files found under the
  ! tree of HOME:
  searchdir "HOME:","listfile";
ENDPROC
```

Dieses Programm durchläuft die Verzeichnisstruktur unter "`HOME`:" und ruft für jede gefundene Datei die Prozedur `listfile` auf. Die Angabe `searchdir` ist ein generischer Teil, der weder den Beginn der Suche noch die aufzurufende Routine für jede Datei kennt. Sie verwendet `IsFile`, um zu prüfen, ob ein Unterverzeichnis oder eine Datei gefunden wurde, und verwendet den Mechanismus der späten Bindung, um für alle gefundenen Dateien die Prozedur aufzurufen, die in `actionproc` angegeben ist. Die `actionproc`-Routine `listfile` prüft, ob die Datei größer als `1` KB ist.

Syntax

```
FileSize '('
  [ Path ':=' ] < expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.70 FileSize - Abrufen der Größe einer Datei

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.71 FileTimeDnum - Abrufen von Zeitinformationen über eine Datei

Verwendung

`FileTimeDnum` wird verwendet, um den letzten Zeitpunkt der letzten Änderung oder des letzten Zugriffs auf eine Datei oder den Zeitpunkt der letzten Statusänderung einer Datei abzurufen. Die Zeit wird in Sekunden seit 00:00:00 GMT, 1. Jan. 1970 gemessen. Die Zeit wird mit dem Datentyp `dnum` und optional auch mit dem Datentyp `stringdig` zurückgegeben.

Grundlegendes Beispiel

Folgende Beispiele veranschaulichen die Funktion `FileTimeDnum`.

Siehe auch *Weitere Beispiele auf Seite 1308*.

Beispiel 1:

```
IF FileTimeDnum ("HOME:/mymod.mod" \ModifyTime) > ModTimeDnum
     ("mymod") THEN
  UnLoad "HOME:/mymod.mod";
  Load \Dynamic, "HOME:/mymod.mod";
ENDIF
```

Dieses Programm lädt ein Modul erneut, wenn eine neuere Version der Quelldatei vorhanden ist. Es ruft mit `ModTimeDnum` den letzten Ladezeitpunkt des angegebenen Moduls ab und vergleicht ihn mit der `FileTimeDnum ("HOME:/mymod.mod" \ModifyTime)` an der Quelle. Wenn das Modul an der Quelle neuer ist, entfernt das Programm das Modul und lädt es erneut.

Rückgabewert

Datentyp: `dnum`

Die Zeit in Sekunden seit 00:00:00 GMT, 1. Jan. 1970.

Argumente

```
FileTimeDnum ( Path [\ModifyTime] | [\AccessTime] | [\StatCTime]
     [\StrDig])
```

`Path`

Datentyp: `string`

Die angegebene Datei mit einem vollständigen oder relativen Pfad.

`[\ModifyTime]`

Datentyp: `switch`

Zeitpunkt der letzten Änderung.

`[\AccessTime]`

Datentyp: `switch`

Zeitpunkt des letzten Zugriffs (Lesen, Abarbeiten der Änderung).

`[\StatCTime]`

Datentyp: `switch`

Letzte Änderung des Dateistatus (Zugriffsqualifikation).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.71 FileTimeDnum - Abrufen von Zeitinformationen über eine Datei

RobotWare - OS

Fortsetzung

`[\StrDig]`

String Digit

Datentyp: `stringdig`

Um die Dateizeit in einer `stringdig`-Darstellung abzurufen.

Programmabarbeitung

Diese Funktion gibt für die angegebene Datei oder das angegebene Verzeichnis einen numerischen Wert zurück, der die Zeit angibt seit:

- der letzten Änderung,
- dem letzten Zugriff,
- der letzten Statusänderung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Die Datei ist nicht vorhanden.

Weitere Beispiele

Weitere Beispiele für die Funktion `FileTimeDnum` sind unten aufgeführt.

Dies ist ein vollständiges Beispiel, das einen "Alarmservice" für maximal 10 Dateien einrichtet.

```
LOCAL RECORD falert
  string filename;
  dnum ftime;
ENDRECORD

LOCAL VAR falert myfiles[10];
LOCAL VAR num currentpos:=0;
LOCAL VAR intnum timeint;

PROC alertInit(num freq)
  currentpos:=0;
  CONNECT timeint WITH mytrap;
  ITimer freq,timeint;
ENDPROC

LOCAL TRAP mytrap
  VAR num pos:=1;
  WHILE pos <= currentpos DO
    IF FileTimeDnum(myfiles{pos}.filename \ModifyTime) >
         myfiles{pos}.ftime THEN
      TPWrite "The file "+myfiles{pos}.filename+" is changed";
    ENDIF
    pos := pos+1;
  ENDWHILE
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
ENDTRAP

PROC alertNew(string filename)
  currentpos := currentpos+1;
  IF currentpos <= 10 THEN
    myfiles{currentpos}.filename := filename;
    myfiles{currentpos}.ftime := FileTimeDnum (filename
         \ModifyTime);
  ENDIF
ENDPROC

PROC alertFree()
  IDelete timeint;
ENDPROC
```

Syntax

```
FileTimeDnum '('
[Path ':=' ] <expression (IN) of string>
['\' ModifyTime] |
['\' AccessTime] |
['\' StatCTime]
['\' StrDig ':=' <variable (VAR) of stringdig>]')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Zeitpunkt der letzten Änderung eines geladenen Moduls	*ModTimeDnum - Abrufen des Dateiänderungszeitpunkts für das geladene Modul auf Seite 1400*
Zeichenfolge, die nur Ziffern enthält	*stringdig - Zeichenfolge, die nur Ziffern enthält auf Seite 1808*
Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten	*StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten auf Seite 1518*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.72 FSSize - Abrufen der Größe eines Dateisystems

Verwendung

`FSSize` (*File System Size*) wird verwendet, um die Größe des Dateisystems abzurufen, in dem die angegebene Datei gespeichert ist. Die Größe in Byte, Kilobyte oder Megabyte wird als `num` angegeben.

Grundlegendes Beispiel

Das folgende Beispiel veranschaulicht die Funktion `FSSize`.

Siehe auch *Weitere Beispiele auf Seite 1311*.

Beispiel 1:

```
PROC main()
  VAR num totalfsyssize;
  VAR num freefsyssize;
  freefsyssize := FSSize("HOME:/spy.log" \Free);
  totalfsyssize := FSSize("HOME:/spy.log" \Total);
  TPWrite NumToStr(((totalfsyssize -
       freefsyssize)/totalfsyssize)*100,0) +" percent used";
ENDPROC
```

Diese Prozedur gibt den belegten Speicherplatz im Dateisystem `HOME`: Dateisystem als Prozentwert aus.

Rückgabewert

Datentyp: `num`

Die Größe in Byte.

Argumente

```
FSSize (Name [\Total] | [\Free] [\Kbyte] [\Mbyte])
```

`Name`

Datentyp: `string`

Der Name einer Datei im Dateisystem mit dem vollständigen oder relativen Pfad.

`[ \Total ]`

Datentyp: `switch`

Ruft die Gesamtgröße des Speicherplatzes im Dateisystem ab.

`[ \Free ]`

Datentyp: `switch`

Ruft die Größe des freien Speicherplatzes im Dateisystem ab.

`[ \Kbyte ]`

Datentyp: `switch`

Konvertieren Sie die Anzahl der zu lesenden Byte in Kilobyte, teilen Sie die Größe durch 1024.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \Mbyte ]`

Datentyp: `switch`

Konvertieren Sie die Anzahl der zu lesenden Byte in Megabyte, teilen Sie die Größe durch 1.048.576 (1024*1024).

Programmabarbeitung

Diese Funktion gibt einen numerischen Wert zurück, der die Größe des Dateisystems darstellt, in dem sich die angegebene Datei befindet.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Das Dateisystem ist nicht vorhanden.
`ERR_FILESIZE`	Die Größe übersteigt die höchste Ganzzahl für num, nämlich 8388608.

Weitere Beispiele

Weitere Beispiele für die Funktion `FSSize` sind unten aufgeführt.

Beispiel 1:

```
LOCAL VAR intnum timeint;

LOCAL TRAP mytrap
  IF FSSize("HOME:/spy.log" \Free)/FSSize("HOME:/spy.log" \Total)
       <= 0.1 THEN
    TPWrite "The disk is almost full";
    alertFree;
  ENDIF
ENDTRAP

PROC alertInit(num freq)
  CONNECT timeint WITH mytrap;
  ITimer freq,timeint;
ENDPROC

PROC alertFree()
  IDelete timeint;
ENDPROC
```

Dies ist ein vollständiges Beispiel für die Implementierung eines Warnungsservice, der eine Warnung am FlexPendant ausgibt, sobald der verbleibende freie Speicherplatz im Dateisystem „`HOME:`“ weniger als 10 % beträgt.

Syntax

```
FSSize '('
  [ Name ':=' ] < expression (IN) of string>
  [ '\' Total ] | [ '\' Free ]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.72 FSSize - Abrufen der Größe eines Dateisystems

RobotWare - OS

Fortsetzung

```
[ '\' Kbyte ]
[ '\' Mbyte ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.73 GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse

Verwendung

`GetAxisDistance` wird verwendet, um die aktuelle Strecke abzulesen, die die Achse seit dem letzten Zurücksetzen zurückgelegt hat. Wenn die Achse eine Rotationsachse ist, wird die Strecke in Grad angegeben, und wenn die Achse linear ist, wird die Strecke in Metern angegeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GetAxisDistance`.

Beispiel 1:

```
PERS dnum distance;
distance := GetAxisDistance(Track,1);
```

Die Gesamtstrecke, die diese Achse 1 an der mechanischen Einheit `Track` seit dem letzten Zurücksetzen zurückgelegt hat, ist in `distance` gespeichert.

Beispiel 2:

```
PERS dnum distanceLimit := 1000;
PERS dnum remaining;
remaining := distanceLimit – GetAxisDistance(Track,1);
```

Die verbleibende Strecke für Achse 1 an der mechanischem Einheit `Track` ist in `remaining` gespeichert.

Rückgabewert

Datentyp: `dnum`

Der Rückgabewert ist die Strecke in Metern oder Grad, die sich diese Achse seit dem letzten Zurücksetzen bewegt hat.

Argumente

```
GetAxisDistance (MechUnit AxisNo)
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`AxisNo`

Datentyp: `num`

Die Nummer der Achse, für welche die zurückgelegte Strecke gelesen werden soll.

Voraussetzungen

`GetAxisDistance` kann nur einen Wert zurückgeben, wenn die *Service Information System*-Konfiguration ausgeführt wurde.

Beispielkonfiguration

Für den Roboter selbst konfigurieren Sie die *SIS Parameters*.

Für die zusätzlichen Achsen konfigurieren Sie die *SIS Single Parameters*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.73 GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse

RobotWare - OS

Fortsetzung

Konfigurieren Sie den Typ *Robot* für die Verwendung der SIS-Konfiguration mit dem Parameter *use_sis*.

Syntax

```
GetAxisDistance '('
  [ MechUnit ':=' ] < variable (VAR) of mecunit > ','
  [ AxisNo ':=' ] < variable (VAR) of num > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
ResetAxisDistance	*ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse auf Seite 613*
ResetAxisMoveTime	*ResetAxisMoveTime - Zurücksetzen des Bewegungszeitzählers der Achse auf Seite 615*
GetAxisMoveTime	*GetAxisMoveTime - Abrufen des Bewegungszeitzählers der Achse auf Seite 1315*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.74 GetAxisMoveTime - Abrufen des Bewegungszeitzählers der Achse

Verwendung

`GetAxisMoveTime` wird verwendet, um die aktuelle Zeit abzulesen, die sich die Achse seit dem letzten Zurücksetzen bewegt hat.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GetAxisMoveTime`.

Beispiel 1:

```
PERS dnum movetime;
movetime := GetAxisMoveTime(Track,1);
```

Die Gesamtzeit, die sich diese Achse 1 an der mechanischen Einheit `Track` seit dem letzten Zurücksetzen bewegt hat, ist in `movetime` gespeichert.

Beispiel 2:

```
PERS dnum timeLimit := 1000;
PERS dnum remaining;
remaining := timeLimit – GetAxisMoveTime(Track,1);
```

Die verbleibende Zeit für Achse 1 an der mechanischen Einheit `Track` ist in `remaining` gespeichert.

Rückgabewert

Datentyp: `dnum`

Der Rückgabewert ist die Gesamtzeit in Stunden, die sich die Achse seit dem letzten Zurücksetzen bewegt hat.

Argumente

```
GetAxisMoveTime (MechUnit AxisNo)
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`AxisNo`

Datentyp: `num`

Die Nummer der Achse, für die die Bewegungszeit abgelesen werden soll.

Syntax

```
GetAxisMoveTime '('
  [ MechUnit ':=' ] < variable (VAR) of mecunit > ','
  [ AxisNo ':=' ] < variable (VAR) of num > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.74 GetAxisMoveTime - Abrufen des Bewegungszeitzählers der Achse

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
ResetAxisDistance	*ResetAxisDistance - Zurücksetzen der Informationen über die zurückgelegte Strecke für die Achse auf Seite 613*
ResetAxisMoveTime	*ResetAxisMoveTime - Zurücksetzen des Bewegungszeitzählers der Achse auf Seite 615*
GetAxisDistance	*GetAxisDistance - Abrufen des Zählers der zurückgelegten Strecke der Achse auf Seite 1313*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.75 GetMaxNumberOfCyclicBool - Maximale Anzahl der Cyclic bool-Bedingungen abrufen

Verwendung

`GetMaxNumberOfCyclicBool` wird zum Abrufen der maximalen Anzahl von Cyclic bool-Bedingungen verwendet, die zur gleichen Zeit verbunden werden können.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetMaxNumberOfCyclicBool`.

Beispiel 1:

```
VAR num maxno := 0;
maxno := GetMaxNumberOfCyclicBool();
TPWrite "Maximum cyclic bool: " \Num:=maxno;
```

Die maximale Anzahl von Cyclic bool-Bedingungen wird auf dem FlexPendant angezeigt.

Rückgabewert

Datentyp: `num`

Syntax

```
GetMaxNumberOfCyclicBool '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Konfiguration einer Cyclic bool-Bedingung	*SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung auf Seite 722*
Entfernen einer Cyclic bool-Bedingung	*RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung auf Seite 601*
Alle Cyclic bool-Bedingungen entfernen	*RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen auf Seite 599*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.76 GetMecUnitName – Abrufen des Namens der mechanischen Einheit

Verwendung

`GetMecUnitName` ruft den Namen einer mechanischen Einheit mit einer der installierten mechanischen Einheiten als Argument ab. Diese Funktion gibt den Namen der mechanischen Einheit als `string` zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GetMecUnitName`.

Beispiel 1:

```
VAR string mecname;
mecname:= GetMecUnitName(ROB1);
```

`mecname` wird der Wert `"ROB1"` als `string` zugeordnet. Alle mechanischen Einheiten (Datentyp `mecunit`) z. B. `ROB1`, sind im System vordefiniert.

Rückgabewert

Datentyp: `string`

Der Rückgabewert ist der Name der mechanischen Einheit als `string`.

Argumente

```
GetMecUnitName ( MechUnit )
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

`MechUnit` akzeptiert eine der vordefinierten mechanischen Einheiten in der Konfiguration.

Syntax

```
GetMecUnitName '('
  [ MechUnit ':=' ] < variable (VAR) of mecunit > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Mechanische Einheit	*mecunit – Mechanische Einheit auf Seite 1731*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.77 GetModalPayLoadMode – Berechnen des ModalPayLoadMode-Werts

Verwendung

`GetModalPayLoadMode` wird verwendet, um `ModalPayLoadMode` abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GetModalPayLoadMode`.

Beispiel 1:

```
IF GetModalPayloadMode() = 1 THEN
  GripLoad piece1;
  MoveL p1, v1000, fine, gripper;
ELSE
  MoveL p1, v1000, fine, tool2 \TLoad:=gripperpiece1;
ENDIF
```

Lesen Sie den `ModalPayLoadMode`-Wert aus dem System aus und verwenden Sie je nach Wert unterschiedliche Codes, um die in der Bewegungsinstruktion verwendete Last festzulegen.

Rückgabewert

Datentyp: `num`

Der Rückgabewert ist die `ModalPayLoadMode`-Einstellung als `num`.

Syntax

```
GetModalPayloadMode '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Systemparameter *ModalPayLoadMode* zur Aktivierung und Deaktivierung der Nutzlast.	*Technisches Referenzhandbuch - Systemparameter*
Verwendung der Nutzlast in Bewegungsinstruktionen.	*MoveL - Lineare Bewegung des Roboters auf Seite 448*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.78 GetMotorTorque – Liest das aktuelle Motordrehmoment.

Verwendung

`GetMotorTorque` wird verwendet, um das aktuelle Drehmoment des Roboters und die externen Achsen der Motoren abzulesen.

`GetMotorTorque` wird vorzugsweise verwendet, um festzustellen, ob ein Servo-Greifer eine Last hält oder nicht.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GetMotorTorque`.

Siehe auch *Weitere Beispiele auf Seite 1321*.

Beispiel 1:

```
VAR num motor_torque2;
motor_torque2 := GetMotorTorque(2);
```

Das aktuelle Motordrehmoment der zweiten Roboterachse wird in `motor_torque2` gespeichert.

Rückgabewert

Datentyp: `num`

Das aktuelle Motordrehmoment Newtonmeter (Nm) der angegebenen Roboterachse oder der externen Achsen.

Argumente

```
GetMotorTorque [\MecUnit ] AxisNo
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, für die eine Achse gelesen werden soll. Wenn dieses Argument weggelassen wird, wird die Achse für den angeschlossenen Roboter gelesen.

`AxisNo`

Datentyp: `num`

Die Nummer der zu lesenden Achse (1 bis 6).

Programmabarbeitung

Die Funktion liest das aktuelle gefilterte Drehmoment des Motors ab, das auf die Motoren des Roboters und die externen Achsen übertragen wird.

Der Wert des Motordrehmoments kann auch als eine Testsignalnummer 2000 angezeigt werden, wenn *TuneMaster* verwendet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AXIS_PAR	Parameter "Achse" in der Funktion ist falsch.

Weitere Beispiele

Folgende Beispiele veranschaulichen die Funktion GetMotorTorque.

Beispiel 1:

```
VAR num torque_value;
torque_value := GetMotorTorque(\MecUnit:=STN1, 1);
```

Das aktuelle Motordrehmoment der ersten Achse von STN1 wird in torque_value gespeichert.

Beispiel 2:

```
VAR num pre_grip_torque;
VAR num post_grip_torque;
..
MoveJ p10, v1000, fine, Gripper;
! Read the torque for axis 5 before gripping the piece
pre_grip_torque:=GetMotorTorque(5);
! Grip the piece
grip_piece;
! Read the torque for axis 5 after gripping the piece
post_grip_torque:=GetMotorTorque(5);
! Compare torque for axis 5 before and after gripping the piece
piece_gripped:=check_gripped_piece(pre_grip_torque,
     post_grip_torque);
IF piece_gripped = TRUE THEN
  GripLoad piece1;
ELSE
  TPWrite "Failed to grip the piece";
  Stop;
ENDIF
..
```

Das aktuelle Drehmoment der Achse 5 des Roboters wird gelesen, bevor das Stück gegriffen wird. Das Drehmoment wird erneut ausgelesen und verglichen, um festzustellen, ob tatsächlich eine zusätzliche Last am Greifer besteht.

Einschränkungen

Das Ergebnis von GetMotorTorque variiert in Abhängigkeit von der Getriebereibung, Motortemperatur etc. Zwei Messungen in der gleichen Position können differieren. Beispiel: Die Getriebetemperatur kann die Reibung ändern und damit auch das Ergebnis.

Die oben beschriebenen Einschränkungen können es unmöglich machen, sehr kleine Änderungen im Drehmoment zu erkennen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.78 GetMotorTorque – Liest das aktuelle Motordrehmoment.

RobotWare - OS

Fortsetzung

Es ist nur möglich, das aktuelle Drehmoment der mechanischen Einheiten zu lesen, die vom aktuellen Programmtask gesteuert werden. Für einen Nicht-Bewegungstask ist es möglich, das Drehmoment der mechanischen Einheiten zu lesen, die vom verknüpften Bewegungstask gesteuert werden.

Syntax

```
GetMotorTorque '('
  ['\' MecUnit ':=' < variable (VAR) of mecunit> ',']
  [AxisNo ':=' ] < expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Liest die aktuellen Motorwinkel.	*ReadMotor - Liest die aktuellen Motorwinkel. auf Seite 1462*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.79 GetNextCyclicBool - Die Namen aller Cyclic bools abrufen

Verwendung

`GetNextCyclicBool` dient zum Abrufen der Namen aller Cyclic bools.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetNextCyclicBool`.

Beispiel 1:

```
VAR num listno := 0;
VAR string name;
...
WHILE GetNextCyclicBool(listno, name) DO
  TPWrite "Cyclic bool: "+name;
  ! listno := listno + 1 is done by GetNextCyclicBool
ENDWHILE
```

Die Namen aller verbundenen Cyclic bools in dem System werden auf dem FlexPendant angezeigt.

Beispiel 2:

```
PERS bool cyclicflag1;
TASK PERS bool cyclicflag2;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  SetupCyclicBool cyclicflag2, di3=1 AND do4=0;
  WHILE GetNextCyclicBool(listno, name) DO
    TPWrite name;
    ! listno := listno + 1 is done by GetNextCyclicBool
  ENDWHILE
  ...
```

`cyclicflag1` und `T_ROB1/cyclicflag1` werden auf dem FlexPendant angezeigt, wenn der RAPID-Code in der RAPID-Task T_ROB1 abgearbeitet wird.

Rückgabewert

Datentyp: `bool`

Der Rückgabewert ist `TRUE`, wenn ein Cyclic bool-Name gefunden wurde, sonst ist er `FALSE`.

Argumente

```
GetNextCyclicBool(ListNumber Name)
```

`ListNumber`

Datentyp: `num`

Gibt an, welche Cyclic bool in der internen Systemliste mit Cyclic bool-Namen abgerufen werden sollen. Bei der Rückgabe wird der Wert dieser Variablen vom System immer um 1 erhöht, um den Zugriff auf die nächsten Cyclic bool-Namen in der Liste zu vereinfachen. Der erste Cyclic bool-Name in der Liste hat den Index 0.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.79 GetNextCyclicBool - Die Namen aller Cyclic bools abrufen

RobotWare - OS

Fortsetzung

`Name`

Datentyp: `string`

Der Name der persistenten Cyclic bool-Variable. Falls die persistente Variable als eine `TASK PERS` definiert wird, ist der erhaltene Name "TASK-Name /Name der persistenten booleschen Variable".

Syntax

```
GetNextCyclicBool '('
  [ ListNumber ':=' ] < variable (VAR) of num> ','
  [ Name ':=' ] < variable (VAR) of string>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Prüfen, ob eine persistente Variable eine Cyclic bool ist	*IsCyclicBool - Prüft, ob eine persistente Variable eine Cyclic bool ist auf Seite 1369*
Konfiguration einer Cyclic bool-Bedingung	*SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung auf Seite 722*
Entfernen einer Cyclic bool-Bedingung	*RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung auf Seite 601*
Alle Cyclic bool-Bedingungen entfernen	*RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen auf Seite 599*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.80 GetNextMechUnit – Abrufen des Namens und der Daten für mechanische Einheiten

Verwendung

`GetNextMechUnit` (*Get Next Mechanical Unit* wird verwendet, um den Namen von mechanischen Einheiten im Robotersystem abzurufen. Außer dem Namen der mechanischen Einheit können verschiedene optionale Eigenschaften der mechanischen Einheit abgerufen werden.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetNextMechUnit`.

Siehe auch *Weitere Beispiele auf Seite 1326*.

Beispiel 1:

```
VAR num listno := 0;
VAR string name := "";

TPWrite "List of mechanical units:";
WHILE GetNextMechUnit(listno, name) DO
  TPWrite name;
  ! listno := listno + 1 is done by GetNextMechUnit
ENDWHILE
```

Die Namen aller mechanischen Einheiten, die im System vorhanden sind, werden am FlexPendant angezeigt.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn eine mechanische Einheit gefunden wurde; sonst `FALSE`.

Argumente

```
GetNextMechUnit( ListNumber UnitName [\MecRef] [\TCPRob] [\NoOfAxes]
     [\MecTaskNo] [\MotPlanNo] [\Active] [\DriveModule]
     [\OKToDeact])
```

`ListNumber`

Datentyp: `num`

Gibt an, welche Einträge in der internen Systemliste mechanischer Einheiten abgerufen werden sollen. Bei der Rückgabe wird der Wert dieser Variablen vom System immer um 1 erhöht, um den Zugriff auf die nächste Einheit in der Liste zu vereinfachen. Die erste mechanische Einheit in der Liste hat den Index 0.

`UnitName`

Datentyp: `string`

Name der mechanischen Einheit

`[\MecRef]`

Datentyp: `mecunit`

Der Systemverweis auf die mechanische Einheit.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.80 GetNextMechUnit – Abrufen des Namens und der Daten für mechanische Einheiten
RobotWare - OS
Fortsetzung

[\TCPRob]

Datentyp: bool

TRUE, wenn die mechanische Einheit ein TCP-Roboter ist, andernfalls FALSE.

[\NoOfAxes]

Datentyp: num

Die Anzahl der Achsen für die mechanische Einheit. Ganzzahliger Wert.

[\MecTaskNo]

Datentyp: num

Die Nummer der Programmtask, die die mechanische Einheit steuert. Ganzzahliger Wert im Bereich 1-20. Wenn keine Steuerung durch eine Programmtask erfolgt, wird -1 zurückgegeben.

Diese tatsächliche Verbindung wird im Domänencontroller der Systemparameter (kann in einigen Systemen zur Laufzeit neu definiert werden) definiert.

[\MotPlanNo]

Datentyp: num

Die Nummer der Bewegungsplanung, die die mechanische Einheit steuert. Ganzzahliger Wert im Bereich 1-6. Wenn keine Steuerung durch eine Bewegungsplanung erfolgt, wird -1 zurückgegeben.

Diese Verbindung wird im Domänencontroller der Systemparameter definiert.

[\Active]

Datentyp: bool

TRUE, wenn die mechanische Einheit aktiv ist, andernfalls FALSE.

[\DriveModule]

Datentyp: num

Von dieser mechanischen Einheit wird die Drive Module-Nummer 1 - 4 verwendet.

[\OKToDeact]

Datentyp: bool

Gibt TRUE zurück, wenn das Deaktivieren der mechanischen Einheit über das RAPID-Programm zulässig ist.

Weitere Beispiele

Weitere Beispiele für die Instruktion GetNextMechUnit sind unten aufgeführt.

Beispiel 1:

```
VAR num listno := 4;
VAR string name := "";
VAR bool found := FALSE;

found := GetNextMechUnit (listno, name);
```

Wenn found auf TRUE gesetzt ist, befindet sich der Name für die Nummer der mechanischen Einheit in der 4 in der Variablen name, andernfalls enthält name nur eine leere Zeichenfolge.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
GetNextMechUnit '('
  [ ListNumber ':=' ] < variable (VAR) of num> ','
  [ UnitName ':=' ] < variable (VAR) of string> ','
  [ '\' MecRef ':=' < variable (VAR) of mecunit> ]
  [ '\' TCPRob ':=' < variable (VAR) of bool> ]
  [ '\' NoOfAxes ':=' < variable (VAR) of num> ]
  [ '\' MecTaskNo ':=' < variable (VAR) of num> ]
  [ '\' MotPlanNo ':=' < variable (VAR) of num> ]
  [ '\' Active ':=' < variable (VAR) of bool>]
  [ '\' DriveModule ':=' < variable (VAR) of num>]
  [ '\' OKToDeact ':=' < variable (VAR) of bool>]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Mechanische Einheit	*mecunit – Mechanische Einheit auf Seite 1731*
Aktivieren/Deaktivieren mechanischer Einheiten	*ActUnit - Aktivieren einer mechanischen Einheit auf Seite 30* *DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.81 GetNextSym - Abrufen des nächsten passenden Symbols

Verwendung

`GetNextSym` (*Get Next Symbol*) wird zusammen mit `SetDataSearch` verwendet, um Datenobjekte aus dem System abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GetNextSym`.

Beispiel 1:

```
VAR datapos block;
VAR string name;
VAR bool truevar:=TRUE;
...
SetDataSearch "bool" \Object:="my.*" \InMod:="mymod"\LocalSym;
WHILE GetNextSym(name,block) DO
  SetDataVal name\Block:=block,truevar;
ENDWHILE
```

Dieser Vorgang setzt alle lokalen `bool`-Datenobjekte, die mit `my` beginnen, im Modul `mymod` auf `TRUE`.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn ein neues Objekt abgerufen wurde; der Objektname und der eingeschlossene Block werden dann in den Argumenten zurückgegeben.

`FALSE`, wenn keine passenden Objekte vorhanden sind.

Argumente

```
GetNextSym (Object Block [\Recursive])
```

`Object`

Datentyp: `string`

Variable (`VAR` oder `PERS`) zum Speichern des Namens des Datenobjekts, das abgerufen wird.

`Block`

Datentyp: `datapos`

Der eingeschlossene Block für das Objekt.

`[ \Recursive ]`

Datentyp: `switch`

Damit wird die Suche im anschließenden Block fortgesetzt, wenn z. B. der Suchvorgang auf Taskebene begonnen hat, werden auch Module und Routinen unter dem Task durchsucht.

Syntax

```
GetNextSym '('
  [ Object ':=' ] < variable or persistent (INOUT) of string > ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[ Block ':='] <variable (VAR) of datapos>
['\' Recursive ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Definieren eines Symbols, das in einem Suchvorgang gesetzt wird	*SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird auf Seite 702*
Abrufen des Werts eines Datenobjekts	*GetDataVal - Abrufen des Werts eines Datenobjekts auf Seite 228*
Setzen des Werts eines Datenobjekts	*SetDataVal - Setzen des Werts eines Datenobjekts auf Seite 707*
Setzen des Werts vieler Datenobjekte	*SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge auf Seite 697*
Zugehöriger Datentyp `datapos`	*datapos – Umschließender Block für ein Datenobjekt auf Seite 1687*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.82 GetNumberOfCyclicBool - Anzahl der Cyclic bool-Bedingungen abrufen

Verwendung

`GetNumberOfCyclicBool` wird zum Abrufen der Anzahl der verbundenen Cyclic bool-Bedingungen verwendet.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetNumberOfCyclicBool`.

Beispiel 1:

```
VAR num listno := 0;
listno := GetNumberOfCyclicBool();
TPWrite "Connected Cyclic bool: " \Num:=listno;
```

Die Anzahl von verbundenen Cyclic bool-Bedingungen wird auf dem FlexPendant angezeigt.

Rückgabewert

Datentyp: `num`

Syntax

```
GetNumberOfCyclicBool '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Konfiguration einer Cyclic bool-Bedingung	*SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung auf Seite 722*
Entfernen einer Cyclic bool-Bedingung	*RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung auf Seite 601*
Alle Cyclic bool-Bedingungen entfernen	*RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen auf Seite 599*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.83 GetServiceInfo - Abrufen von Informationen über das System

Verwendung

`GetServiceInfo` wird verwendet, um Serviceinformationen aus dem System zu lesen. Diese Funktion gibt die Serviceinformationen als `string` zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GetServiceInfo`.

Siehe auch *Weitere Beispiele auf Seite 1332*.

Beispiel 1:

```
VAR string mystring;
VAR num mynum;
IF TaskRunRob() THEN
  mystring:=GetServiceInfo(ROB_ID \DutyTimeCnt);
  IF StrToVal(mystring, mynum) = FALSE THEN
    TPWrite "Conversion failed!";
    Stop;
  ENDIF
ENDIF
```

Wenn die Task einen Roboter steuert, verwenden Sie die vordefinierte Variable `ROB_ID`, um den Betriebsstundenzähler abzulesen. Konvertieren Sie dann den Zeichenfolgenwert in einen numerischen Wert.

Rückgabewert

Datentyp: `string`

Der Wert der Serviceinformationen für die angegebene mechanische Einheit. Mehr Informationen über die Rückgabewerte siehe nachfolgend unter *Argumente*.

Argumente

```
GetServiceInfo (MechUnit [\DutyTimeCnt])
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, für die Informationen abgerufen werden sollen.

`[\DutyTimeCnt]`

Duty Time Counter

Datentyp: `switch`

Gibt den Betriebsstundenzähler für die mechanische Einheit zurück, die im Argument `MechUnit` verwendet wird. Wenn diese Option in der virtuellen Steuerung verwendet wird, wird eine Zeichenfolge zurückgegeben, die „0“ enthält.

Der Betriebsstundenzähler ist der Wert in Stunden, in der sich die mechanische Einheit im Zustand „Motors on“ befand und die Bremsen gelöst waren.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.83 GetServiceInfo - Abrufen von Informationen über das System
RobotWare - OS
Fortsetzung

Programmabarbeitung

Die Serviceinformationen werden für den verwendeten optionalen Parameter abgelesen.

Weitere Beispiele

Weitere Beispiele für die Verwendung der Funktion `GetServiceInfo` sind unten aufgeführt.

Beispiel 1:

```
VAR string mystring;
mystring:=GetServiceInfo(ROB_1 \DutyTimeCnt);
TPWrite "DutyTimeCnt for ROB_1: " + mystring;
mystring:=GetServiceInfo(ROB_2 \DutyTimeCnt);
TPWrite "DutyTimeCnt for ROB_2: " + mystring;
mystring:=GetServiceInfo(INTERCH \DutyTimeCnt);
TPWrite "DutyTimeCnt for INTERCH: " + mystring;
mystring:=GetServiceInfo(STN_1 \DutyTimeCnt);
TPWrite "DutyTimeCnt for STN_1: " + mystring;
mystring:=GetServiceInfo(STN_2 \DutyTimeCnt);
TPWrite "DutyTimeCnt for STN_2: " + mystring;
```

Ruft Informationen über den Betriebsstundenzähler für alle mechanischen Einheiten in einem MultiMove-System ab und schreibt die Werte auf das FlexPendant.

Syntax

```
GetServiceInfo '('
  [MechUnit ':=' ] <variable (VAR) of mecunit> ','
  ['\' DutyTimeCnt] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Mechanische Einheit	*mecunit – Mechanische Einheit auf Seite 1731*.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.84 GetSignalOrigin - Abruf von Informationen zur Herkunft eines E/A-Signals.

Verwendung

`GetSignalOrigin` wird verwendet, um Informationen zur Herkunft eines E/A-Signals zu erhalten.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GetSignalOrigin`.

Beispiel 1:

```
VAR signalorigin myorig;
VAR string signalname;
...
myorig:=GetSignalOrigin(mysignal, signalname);
IF myorig = SIGORIG_NONE THEN
  TPWrite "Signal cannot be used. AliasIO needed.";
ELSEIF myorig = SIGORIG_CFG THEN
  TPWrite "Signal "+signalname+" is defined in I/O configuration.";
ELSEIF myorig = SIGORIG_ALIAS THEN
  TPWrite "Signal is declared in RAPID.";
  TPWrite "Name according to the I/O configuration: "+signalname;
ENDIF
```

Der oben stehende Code kann verwendet werden, um die Herkunft des als `mysignal` bezeichneten Signals festzustellen.

Rückgabewert

Datentyp: `signalorigin`

`signalorigin`, wie in der Tabelle unten beschrieben.

Rückgabe-wert	Symbolische Konstante	Kommentar
0	`SIGORIG_NONE`	E/A Die E/A-Signalvariable wird in RAPID erklärt und weist keine Alias-Kupplung auf.
1	`SIGORIG_CFG`	Das Signal ist in der E/A-Konfiguration konfiguriert.
2	`SIGORIG_ALIAS`	Die E/A-Signalvariable wird in RAPID erklärt und weist eine Alias-Verkopplung zu einem in einer E/A-Konfiguration konfigurierten Signal auf.

Argumente

```
GetSignalOrigin Signal SignalName
```

`Signal`

Datentyp: `signalxx`

Der Signalname. Muss vom Datentyp `signaldo`, `signaldi`, `signalgo`, `signalgi`, `signalao`, oder `signalai`. sein.

`SignalName`

Datentyp: `string`

Der Signalname gemäß der E/A-Konfiguration oder leerer String.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.84 GetSignalOrigin - Abruf von Informationen zur Herkunft eines E/A-Signals.

RobotWare - OS

Fortsetzung

Programmabarbeitung

Die Funktion gibt einen der folgenden vordefinierten Signalursprünge zurück: `SIGORIG_NONE`, `SIGORIG_CFG` oder `SIGORIG_ALIAS`.

Wenn `SIGORIG_NONE` zurückgegeben wird, besteht `SignalName` aus einem leeren String.

Wenn `SIGORIG_CFG` oder `SIGORIG_ALIAS` zurückgegeben wird, enthält das Argument `SignalName` den E/A-Signalnamen entsprechend der E/A-Konfiguration.

`GetSignalOrigin` kann bei allgemeinen Programmen verwendet werden, um zu überprüfen, ob ein Signal eine Alias-Verkopplung hat und ob es sich um eine Verkopplung zum richtigen physikalischen E/A-Signal handelt.

Syntax

```
GetSignalOrigin
  [Signal ':='] <variable (VAR) of anytype>','
  [SignalName ':='] <variable (VAR) of string>';'
```

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*
Definieren eines E/A-Signals mit Aliasname	*AliasIO - Definieren eines E/A-Signals mit Aliasname auf Seite 37*
Zurücksetzen eines E/A-Signals mit Aliasnamen	*AliasIOReset – Zurücksetzen eines E/A-Signals mit Aliasnamen auf Seite 40*
Datentyp `signalorigin`	*signalorigin - Gibt die Herkunft des E/A-Signals an auf Seite 1789*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.85 GetSysInfo - Abrufen von Informationen über das System

Verwendung

`GetSysInfo` wird verwendet, um Informationen über das System zu lesen. Die verfügbaren Informationen sind z. B. Seriennummer, Softwareversion, Name der Softwareversion, Robotertyp, Steuerungs-ID, WAN-IP-Adresse, Steuerungssprache oder Systemname.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetSysInfo`.

Beispiel 1:

```
VAR string serial;
VAR string version;
VAR string versionname;
VAR string rtype;
VAR string cid;
VAR string lanip;
VAR string clang;
VAR string sysname;
serial := GetSysInfo(\SerialNo);
version := GetSysInfo(\SWVersion);
versionname := GetSysInfo(\SWVersionName);
rtype := GetSysInfo(\RobotType);
cid := GetSysInfo(\CtrlId);
lanip := GetSysInfo(\LanIp);
clang := GetSysInfo(\CtrlLang);
sysname := GetSysInfo(\SystemName);
```

Beispiele für zurückgegebene Zeichenfolgen:

Beschreibung	Variable	Rückgabewert
Seriennummer	`serial`	24-12345
Softwareversion	`version`	ROBOTWARE_6.03.xxxx
Name der Softwareversion	`versionname`	6.03.00.00
Roboternummer	`rtype`	IRB 2400-16/1.5 Type A
Steuerungs-ID	`cid`	MyRobot
WAN-IP-Adresse	`lanip`	192.168.8.103
Steuerungssprache	`clang`	en
Aktives System	`sysname`	MySystem

Rückgabewert

Datentyp: `string`

Der Rückgabewert ist eine Zeichenfolge mit Seriennummer, Softwareversion, Name der Softwareversion, Robotertyp, Steuerungs-ID, WAN-IP-Adresse, Steuerungssprache und Systemname. Lesen Sie mehr über die Rückgabewerte in *Argumente auf Seite 1336*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.85 GetSysInfo - Abrufen von Informationen über das System
RobotWare - OS
Fortsetzung

Argumente

```
GetSysInfo ([\SerialNo] | [\SWVersion] | [\SWVersionName] |
     [\RobotType] | [\CtrlId] | [\LanIp] | [\CtrlLang] |
     [\SystemName])
```

Mindestens eines der Argumente muss vorhanden sein.

`[\SerialNo]`

Serial Number

Datentyp: `switch`

Gibt die Seriennummer zurück.

`[\SWVersion]`

Software Version

Datentyp: `switch`

Gibt die Medienversion von RobotWare aus, die im Ordner *PRODUCTS* installiert ist.

`[\SWVersionName]`

Software Version Name

Datentyp: `switch`

Gibt den Anzeigenamen der Medienversion von RobotWare zurück.

`[\RobotType]`

Datentyp: `switch`

Gibt den Robotertyp in der aktuellen oder verbundenen Task zurück. Wenn die mechanische Einheit kein TCP-Roboter ist, wird ein Gedankenstrich (-) zurückgegeben.

`[\CtrlId]`

Controller ID

Datentyp: `switch`

Gibt die Steuerungs-ID zurück. Gibt eine leere Zeichenfolge zurück, wenn keine Steuerungs-ID angegeben wurde. Die Zeichenfolge `VC` wird zurückgegeben, wenn sie in einer virtuellen Steuerung verwendet wird.

`[\LanIp]`

Lan Ip address

Datentyp: `switch`

Gibt die WAN-IP-Adresse für die Steuerung zurück. Die Zeichenfolge `VC` wird zurückgegeben, wenn sie in einer virtuellen Steuerung verwendet wird. Wenn keine WAN-IP-Adresse im System konfiguriert ist, wird eine leere Zeichenfolge zurückgegeben.

`[\CtrlLang]`

Controller Language

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Gibt die in der Steuerung verwendete Sprache zurück.

Rückgabewert	Sprache
cs	Tschechisch
zh	Chinesisch (Chinesisch vereinfacht, auf dem chinesischen Festland verwendet)
da	Dänisch
nl	Niederländisch
en	Englisch
fi	Finnisch
fr	Französisch
de	Deutsch
hu	Ungarisch
it	Italienisch
ja	Japanisch
ko	Koreanisch
pl	Polnisch
pt	Portugiesisch (Brasilianisches Portugiesisch)
ro	Rumänisch
ru	Russisch
sl	Slowenisch
es	Spanisch
sv	Schwedisch
tr	Türkisch

`[\SystemName]`

Datentyp: `switch`

Gibt den Namen des aktiven Systems zurück.

Syntax

```
GetSysInfo '('
  ['\'SerialNo]
  |['\' SWVersion]
  |['\' SWVersionName]
  |['\' RobotType]
  |['\' CtrlId]
  |['\' LanIp]
  |['\' CtrlLang]
  |['\' SystemName]')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Testen der Systemidentität	*IsSysId - Testen der Systemidentität auf Seite 1387*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.86 GetTaskName - Abrufen des Namens und der Nummer der aktuellen Task

Verwendung

`GetTaskName` wird verwendet, um die Identität der aktuellen Programmtask samt ihrem Namen und ihrer Nummer abzurufen.

Es ist auch möglich, den Namen und die Nummer einer *Nicht-Bewegungstask* abzurufen, wenn sie mit einer *Bewegungstask* verbunden ist. Für ein *MultiMove-System* definieren die Systemparameter *Controller/Tasks/Use Mechanical Unit Group* die verbundene *Bewegungstask* und in einem Basissystem ist die Haupttask von jeder anderen Task immer die verbundene *Bewegungstask*.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GetTaskName`.

Beispiel 1:

```
VAR string taskname;
...
taskname := GetTaskName();
```

Der aktuelle Taskname wird in der Variablen `taskname` zurückgegeben.

Beispiel 2:

```
VAR string taskname;
VAR num taskno;
...
taskname := GetTaskName(\TaskNo:=taskno);
```

Der aktuelle Taskname wird in der Variablen `taskname` zurückgegeben. Die ganzzahlige Identität des Task wird in der Variablen `taskno` gespeichert.

Beispiel 3

```
VAR string taskname;
VAR num taskno;
...
taskname := GetTaskName(\MecTaskNo:=taskno);
```

Wenn die aktuelle Task *keine Bewegungstask* ist, wird der Name der verbundenen Bewegungstask in der Variablen `taskname` zurückgegeben. Die numerische Identität der verbundenen Bewegungstask wird in der Variablen `taskno` gespeichert.

Wenn die aktuelle Task mehrere mechanische Einheiten steuert, wird der aktuelle Taskname in der Variablen `taskname` zurückgegeben. Die numerische Identität der Task wird in der Variablen `taskno` gespeichert.

Rückgabewert

Datentyp: `string`

Der Name der Task, in der die Funktion abgearbeitet wird, oder der Name der verbundenen Bewegungstask.

Argumente

```
GetTaskName ( [\TaskNo] | [\MecTaskNo] )
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\TaskNo]`

Datentyp: `num`

Gibt den Namen der aktuellen Task zurück (gleiche Funktionalität, wenn keiner der Switches `\TaskNo` oder `\MecTaskNo` verwendet wird). Ruft außerdem die Identität der aktuellen Task als ganzzahligen Wert ab. Die zurückgegebenen Zahlen liegen im Bereich 1-20.

`[\MecTaskNo]`

Datentyp: `num`

Gibt den Namen der verbundenen Bewegungstask oder der aktuellen Bewegungstask zurück. Ruft außerdem die Identität der verbundenen oder aktuellen Bewegungstask als ganzzahligen Wert ab. Die zurückgegebenen Zahlen liegen im Bereich 1-20.

Syntax

```
GetTaskName '('
  [ \TaskNo ':=' ] < variable (VAR) of num >
  [ \MecTaskNo ':=' ] < variable (VAR) of num > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Prüfen, ob die Task einen TCP-Roboter steuert	*TaskRunRob - Überprüfen, ob die Task einen Roboter steuert auf Seite 1542*
Multitasking	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt*RAPID Overview - RAPID summary Multitasking* *Technisches Referenzhandbuch - RAPID Overview*, Abschnitt*Basic characteristics - Multitasking*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.87 GetTime - Lesen der aktuellen Zeit als numerischer Wert

Verwendung

`GetTime` wird verwendet, um eine angegebene Komponente der aktuellen Systemzeit als numerischen Wert zu lesen.

`GetTime` kann verwendet werden zum:

- Ausführen einer Aktion zu einem bestimmten Zeitpunkt,
- Ausführen bestimmter Aktivitäten an einem Wochentag,
- Unterlassen bestimmter Aktivitäten am Wochenende,
- Abhängig von der Tageszeit unterschiedliche Reaktion auf Fehler.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GetTime`.

Siehe auch *Weitere Beispiele auf Seite 1341*.

Beispiel 1:

```
hour := GetTime(\Hour);
```

Die aktuelle Stunde wird in der Variablen `hour` gespeichert.

Rückgabewert

Datentyp: `num`

Eine der vier unten angegebenen Zeitkomponenten.

Argument

```
GetTime ( [\WDay] | [\Hour] | [\Min] | [\Sec] )
```

[\WDay]

Datentyp: `switch`

Gibt den aktuellen Wochentag zurück. Bereich: 1 bis 7 (Montag bis Sonntag).

[\Hour]

Datentyp: `switch`

Gibt die aktuelle Stunde zurück. Bereich: 0 bis 23.

[\Min]

Datentyp: `switch`

Gibt die aktuelle Minute zurück. Bereich: 0 bis 59.

[\Sec]

Datentyp: `switch`

Gibt die aktuelle Sekunde zurück. Bereich: 0 bis 59.

Eines der Argumente muss angegeben sein, andernfalls stoppt die Programmabarbeitung mit einer Fehlermeldung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Weitere Beispiele für die Funktion `GetTime` sind unten aufgeführt.

Beispiel 1:

```
weekday := GetTime(\WDay);
hour := GetTime(\Hour);
IF weekday < 6 AND hour >6 AND hour < 16 THEN
  production;
ELSE
  maintenance;
ENDIF
```

An einem Wochentag zwischen 7:00 und 15:59 führt der Roboter die Produktion aus. Zu allen anderen Zeiten befindet er sich im Wartungsmodus.

Syntax

```
GetTime '('
['\' WDay ]
| [ '\' Hour ]
| [ '\' Min ]
| [ '\' Sec ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Zeit- und Datums-Instruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - System & Time*
Einstellen der Systemuhr	*Bedienungsanleitung - IRC5 mit FlexPendant*, Abschnitt *Changing FlexPendant settings*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.88 GetTorqueMargin - Liest die geringste Drehmomentspanne ab

Verwendung

`GetTorqueMargin` wird verwendet, um die geringste Drehmomentspanne seit der Ausführung von `ResetTorqueMargin` abzulesen und kann sowohl für Roboter- als auch für externe Achsen verwendet werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GetTorqueMargin`.

Beispiel 1:

```
VAR num torque_margin;
ResetTorqueMargin \AxisNo:=5;
! Insert Program Here
! ...
! ...
torque_margin := GetTorqueMargin (5);
```

Die geringste Drehmomentspanne seit der letzten Ausführung von `ResetTorqueMargin 5` von der Achse 5 des Roboters wird in `torque_margin` gespeichert.

Beispiel 2:

```
VAR num torque_margin1;
VAR num torque_margin2;
VAR num torque_margin3;

ResetTorqueMargin \AxisNo:=5;
! Insert Program Here
! ...
! ...
torque_margin1 := GetTorqueMargin (5);
ResetTorqueMargin 5;
! Change arm config
! Insert Program Here
! ...
! ...
torque_margin2 := GetTorqueMargin (5);
ResetTorqueMargin 5;
! Change arm config
! Insert Program Here
! ...
! ...
torque_margin3 := GetTorqueMargin (5);

! compare torque_margin1, torque_margin2, torque_margin3 etc
```

Rückgabewert

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die geringste Drehmomentspanne in % des verfügbaren Drehmoments seit Ausführung von `ResetTorqueMargin`.

Argumente

```
GetTorqueMargin [\MecUnit ] AxisNo
```

`[\MecUnit ]`

Datentyp: `mecunit`

Der Name der mechanischen Einheit, für die eine Achse gelesen werden soll. Wenn dieses Argument weggelassen wird, wird die Achse für den angeschlossenen Roboter gelesen.

`AxisNo`

Datentyp: `num`

Die Nummer der zu lesenden Achse (1 bis 6).

Programmabarbeitung

Die Funktion liest den geringsten Motordrehmoment von der letzten `ResetTorqueMargin`.

Die Motordrehspanne kann auch über das Testsignal 4040 abgelesen werden. (Dieses Signal zeigt immer die aktuelle Drehmomentspanne an.)

Einschränkungen

Wenn externe Kräfte auf den Roboter einwirken, muss darauf geachtet werden, dass die Struktur nicht überlastet wird. Diese Methode liest nur den Drehmomentwert der Motoren.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_AXIS_PAR`	Parameter "Achse" in der Funktion ist falsch.

Syntax

```
GetTorqueMargin '('
  [ '\' MechUnit ':=' ] < variable (VAR) of mecunit > ','
  [ '\' AxisNo ':=' < expression (IN) of num > ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Zurückstellen der Drehmomentspanne	*ResetTorqueMargin - Die geringste Drehmomentspanne zurückstellen auf Seite 619*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.89 GetTSPStatus – Rufen Sie den aktuellen Status des Taskauswahlfeldes ab

Verwendung

`GetTSPStatus` wird verwendet, um zu überprüfen, ob die Task im **Taskauswahlfeld** auf dem FlexPendant aktiviert oder deaktiviert ist.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetTSPStatus`.

Beispiel 1:

```
VAR tsp_status tspstatus;
...
tspstatus:=GetTSPStatus("MYTASK");
IF tspstatus >= TSP_NORMAL_UNCHECKED AND tspstatus <=
     TSP_SEMISTATIC_UNCHECKED THEN
  TPWrite "Task MYTASK is unchecked in the Task Selection Panel";
ELSEIF tspstatus >= TSP_NORMAL_CHECKED THEN
  TPWrite "Task MYTASK is checked in the Task Selection Panel";
ELSE
  TPWrite "Task MYTASK is unchecked in TSP due to execution in
       service routine";
ENDIF
```

Überprüfen Sie, ob die Programmtask `MYTASK` im **Taskauswahlfeld** auf dem FlexPendant ausgewählt oder abgewählt ist.

Rückgabewert

Datentyp: `tsp_status`

Der aktuelle Status des Taskauswahlfeldes.

Argumente

```
GetTSPStatus (TaskRef | TaskName)
```

`TaskRef`

Datentyp: `taskid`

Die Programmtask-Identität der Task, die ausgewählt werden soll.

Die vordefinierten Variablen des Datentyps `taskid` sind für alle Programmtasks im System verfügbar.

Die Variablenidentität ist *"taskname"+"Id"*, z. B. ist die Variablenidentität für die `T_ROB1`-Task `T_ROB1Id`.

`TaskName`

Datentyp: `string`

Der Programmtaskname der Task, die ausgewählt werden soll.

Vordefinierte Daten

Folgende vordefinierte, symbolische Konstanten des Typs `tsp_status` können zur Überprüfung des Rückgabewerts verwendet werden:

```
CONST tsp_status TSP_UNCHECKED_RUN_SERV_ROUT := 10;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
CONST tsp_status TSP_NORMAL_UNCHECKED := 11;
CONST tsp_status TSP_STATIC_UNCHECKED := 12;
CONST tsp_status TSP_SEMISTATIC_UNCHECKED := 13;
CONST tsp_status TSP_NORMAL_CHECKED := 14;
CONST tsp_status TSP_STATIC_CHECKED := 15;
CONST tsp_status TSP_SEMISTATIC_CHECKED := 16;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.

Syntax

```
GetTSPStatus '('
  [ TaskRef ':=' ] <variable (VAR) of taskid>
  |[ TaskName ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs tsp_status.

Weitere Informationen

Information über	Siehe
Status des Taskauswahlfeldes	*tsp_status – Status des Taskauswahlfeldes auf Seite 1846*
Prüfen Sie, ob eine normale Task aktiv ist	*TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist auf Seite 1545*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.90 GetUASUserName - Abfrage des Benutzernamens für Benutzer

Verwendung

`GetUASUserName` dient zum Abrufen des Benutzernamens des derzeit über FlexPendant angemeldeten Benutzers.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetUASUserName`.

Beispiel 1:

```
VAR string strUser;
...
strUser := GetUASUserName();
```

Der Name des aktuellen Benutzers wird in der Variablen `strUser` zurückgegeben.

Rückgabewert

Datentyp: `string`

Der Name des aktuell angemeldeten Benutzers.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_TP_NO_CLIENT`	Auf dem FlexPendant ist kein Benutzer angemeldet, d. h. es wird derzeit kein FlexPendant verwendet.

Weitere Beispiele

Folgende Beispiele veranschaulichen die Funktion `GetUASUserName`.

Siehe auch *Weitere Beispiele auf Seite 1346*.

Beispiel 1:

```
VAR string user_name:="No user";
...
IF UIClientExist() = TRUE THEN
  user_name:=GetUASUserName();
ENDIF
```

Der Name des aktuellen Benutzers wird in der Variablen `user_name` zurückgegeben. Bei Betrieb ohne FlexPendant enthält der `user_name` die Zeichenfolge `"No user"`.

Syntax

```
GetUASUserName'(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
System zur Anwender-Autorisierung (UAS)	*Bedienungsanleitung - RobotStudio*
Überprüfen, ob der Benutzer-Client vorhanden ist	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.91 GInput - Liest den Wert des Gruppeneingangssignals.

Verwendung

`GInput` wird verwendet, um den aktuellen Wert einer Gruppe von digitalen Eingangssignalen zu lesen.

> **Hinweis**
>
> Beachten Sie, dass die Funktion `GInput` eine alte Funktion ist, die nicht länger verwendet werden darf. Beachten Sie die Beispiele für eine alternative und empfohlene Art der Programmierung.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GInput`.

Beispiel 1:

```
IF GInput(gi2) = 5 THEN ...
...
IF gi2 = 5 THEN ...
```

Wenn der aktuelle Wert des Signals `gi2` gleich `5` ist, dann `...`

Rückgabewert

Datentyp: `num`

Der aktuelle Wert des Signals (eine positive Ganzzahl).

Der Wert jedes Signals in der Gruppe wird gelesen und als Binärzahl ohne Vorzeichen interpretiert. Diese Binärzahl wird dann in eine Ganzzahl konvertiert.

Der zurückgegebene Wert liegt in einem Bereich, der von der Anzahl der Signale in der Gruppe abhängt.

Anzahl der Signale	Zulässiger Wert
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Anzahl der Signale	Zulässiger Wert
15	0-32767
16	0-65535
17	0-131071
18	0-262143
19	0-524287
20	0-1048575
21	0-2097151
22	0-4194303
23	0-8388607

Argumente

```
GInput (Signal)
```

`Signal`

Datentyp: `signalgi`

Der Name der zu lesenden Signalgruppe.

Syntax

```
GInput '('
  [Signal ':='] <variable (VAR) of signalgi>')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Liest den Wert des Gruppeneingangssignals mit mehr als 23 Bit	*GInputDnum - Liest den Wert des Gruppeneingangssignals. auf Seite 1350*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.92 GInputDnum - Liest den Wert des Gruppeneingangssignals.

Verwendung

`GInputDnum` wird verwendet, um den aktuellen Wert einer Gruppe von digitalen Eingangssignalen zu lesen, die größer als 23 Bit sind.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GInputDnum`.

Beispiel 1:

```
IF GInputDnum(gi2) = 55 THEN ...
```

Wenn der aktuelle Wert des Signals `gi2` gleich `55` ist, dann `...`

Beispiel 2:

```
IF GInputDnum(gi2) = 4294967295 THEN ...
```

Wenn der aktuelle Wert des Signals `gi2` gleich `4294967295` ist, dann `...`

Rückgabewert

Datentyp: `dnum`

Der aktuelle Wert des Signals (eine positive Ganzzahl).

Der Wert jedes Signals in der Gruppe wird gelesen und als Binärzahl ohne Vorzeichen interpretiert. Diese Binärzahl wird dann in eine Ganzzahl konvertiert.

Der zurückgegebene Wert liegt in einem Bereich, der von der Anzahl der Signale in der Gruppe abhängt.

Anzahl der Signale	Zulässiger Wert
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071
18	0-262143

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Anzahl der Signale	Zulässiger Wert
19	0-524287
20	0-1048575
21	0-2097151
22	0-4194303
23	0-8388607
24	0-16777215
25	0-33554431
26	0-67108863
27	0-134217727
28	0-268435455
29	0-536870911
30	0-1073741823
31	0-2147483647
32	0-4294967295

Argumente

```
GInputDnum (Signal)
```

`Signal`

Datentyp: `signalgi`

Der Name der zu lesenden Signalgruppe.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
GInputDnum '('
  [ Signal ':=' ] < variable (VAR) of signalgi > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.92 GInputDnum - Liest den Wert des Gruppeneingangssignals.
RobotWare - OS
Fortsetzung

Weitere Informationen

Information über	Siehe
Lesen des Wertes des Gruppeneingangssignals	*GInput - Liest den Wert des Gruppeneingangssignals. auf Seite 1348*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.93 GOutput - Lesen des Werts von einer Gruppe digitaler Ausgangssignale

Verwendung

`GOutput` wird verwendet, um den aktuellen Wert einer Gruppe von digitalen Ausgangssignalen zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `GOutput`.

Beispiel 1:

```
IF GOutput(go2) = 5 THEN ...
```

Wenn der aktuelle Wert des Signals `go2` gleich `5` ist, dann `...`

Rückgabewert

Datentyp: `num`

Der aktuelle Wert des Signals (eine positive Ganzzahl).

Der Wert jedes Signals in der Gruppe wird gelesen und als Binärzahl ohne Vorzeichen interpretiert. Diese Binärzahl wird dann in eine Ganzzahl konvertiert.

Der zurückgegebene Wert liegt in einem Bereich, der von der Anzahl der Signale in der Gruppe abhängt.

Anzahl der Signale	Zulässiger Wert
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071
18	0-262143
19	0-524287
20	0-1048575

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.93 GOutput - Lesen des Werts von einer Gruppe digitaler Ausgangssignale

RobotWare - OS

Fortsetzung

Anzahl der Signale	Zulässiger Wert
21	0-2097151
22	0-4194303
23	0-8388607

Argumente

```
GOutput (Signal)
```

`Signal`

Datentyp: `signalgo`

Der Name der zu lesenden Signalgruppe.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
GOutput '('
  [ Signal ':=' ] < variable (VAR) of signalgo > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Festlegen einer Gruppe von Ausgangssignalen	*SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen auf Seite 713*
Liest eine Gruppe von Ausgangssignalen.	*GOutputDnum – Lesen des Werts der Gruppe Ausgangssignale auf Seite 1355*
Liest eine Gruppe von Eingangssignalen.	*GInputDnum - Liest den Wert des Gruppeneingangssignals. auf Seite 1350*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.94 GOutputDnum – Lesen des Werts der Gruppe Ausgangssignale

Verwendung

`GOutputDnum` wird verwendet, um den aktuellen Wert einer Gruppe von digitalen Ausgangssignalen zu lesen, die größer als 23 Bit sind.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `GOutputDnum`.

Beispiel 1:

```
IF GOutputDnum(go2) = 55 THEN ...
```

Wenn der aktuelle Wert des Signals `go2` gleich `55` ist, dann `...`

Beispiel 2:

```
IF GOutputDnum(go2) = 4294967295 THEN ...
```

Wenn der aktuelle Wert des Signals `go2` gleich `4294967295` ist, dann `...`

Rückgabewert

Datentyp: `dnum`

Der aktuelle Wert des Signals (eine positive Ganzzahl).

Der Wert jedes Signals in der Gruppe wird gelesen und als Binärzahl ohne Vorzeichen interpretiert. Diese Binärzahl wird dann in eine Ganzzahl konvertiert.

Der zurückgegebene Wert liegt in einem Bereich, der von der Anzahl der Signale in der Gruppe abhängt.

Anzahl der Signale	Zulässiger Wert
1	0-1
2	0-3
3	0-7
4	0-15
5	0-31
6	0-63
7	0-127
8	0-255
9	0-511
10	0-1023
11	0-2047
12	0-4095
13	0-8191
14	0-16383
15	0-32767
16	0-65535
17	0-131071
18	0-262143

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.94 GOutputDnum – Lesen des Werts der Gruppe Ausgangssignale
RobotWare - OS
Fortsetzung

Anzahl der Signale	Zulässiger Wert
19	0-524287
20	0-1048575
21	0-2097151
22	0-4194303
23	0-8388607
24	0-16777215
25	0-33554431
26	0-67108863
27	0-134217727
28	0-268435455
29	0-536870911
30	0-1073741823
31	0-2147483647
32	0-4294967295

Argumente

```
GOutputDnum (Signal)
```

`Signal`

Datentyp: `signalgo`

Der Name der zu lesenden Signalgruppe.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
GOutputDnum '('
  [ Signal ':=' ] < variable (VAR) of signalgo > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Festlegen einer Gruppe von Ausgangssignalen	*SetGO - Ändern des Werts einer Gruppe von digitalen Ausgangssignalen auf Seite 713*l
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Überblick*, Abschnitt *RAPID-Zusammenfassung - Eingangs- und Ausgangssignale*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Überblick*, Abschnitt *Bewegung und E/A-Prinzipien*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.95 HexToDec - Konvertieren von Hexadezimal- in Dezimalformat

Verwendung

`HexToDec` wird verwendet, um die in einer lesbaren Zeichenfolge angegebene Zahl mit der Basis 16 in die Basis 10 zu konvertieren.

Die Eingangszeichenfolge muss aus dem Zeichensatz [0-9, A-F, a-F] erstellt werden.

Diese Routine handhabt Zahlen von 0 bis 9223372036854775807dec bzw. 7FFFFFFFFFFFFFFF hex.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `HexToDec`.

Beispiel 1:

```
VAR string str;

str := HexToDec("5F5E0FF");
```

Die Variable `str` erhält den Wert "99999999".

Rückgabewert

Datentyp: `string`

Die Zeichenfolge, die in eine dezimale Darstellung der im Eingangsparameter angegebenen Zahl konvertiert wurde.

Argumente

```
HexToDec ( Str )
```

`Str`

String

Datentyp: `string`

Die zu konvertierende Zeichenfolge.

Syntax

```
HexToDec '('
  [ Str ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - String functions*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Basic elements*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.96 IndInpos - Unabhängige Achse in Positionsstatus

Verwendung

`IndInpos` wird verwendet, um zu prüfen, ob eine unabhängige Achse die ausgewählte Position erreicht hat.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IndInpos`.

Beispiel 1:

```
IndAMove Station_A,1\ToAbsNum:=90,20;
WaitUntil IndInpos(Station_A,1) = TRUE;
WaitTime 0.2;
```

Warten, bisAchse `1` von `Station_A` in 90-Grad-Position ist.

Rückgabewert

Datentyp: `bool`

Die Tabelle beschreibt die Rückgabewerte von `IndInpos`:

Rückgabewert	Achsenstatus
`TRUE`	In Position und hat null Geschwindigkeit.
`FALSE`	Nicht in Position und/oder hat nicht null Geschwindigkeit.

Argumente

```
IndInpos ( MecUnit Axis )
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.
`ERR_AXIS_IND`	Die Achse befindet sich nicht im unabhängigen Modus.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.96 IndInpos - Unabhängige Achse in Positionsstatus

Independent Axis

Fortsetzung

Einschränkungen

Eine unabhängige Achse, mit der die Instruktion `IndCMove` abgearbeitet wird, gibt immer den Wert `FALSE` zurück, selbst wenn die Geschwindigkeit auf null gesetzt ist.

Eine Wartezeit von 0,2 Sekunden sollte nach der Instruktion hinzugefügt werden, um sicherzustellen, dass der korrekte Status erreicht wurde. Diese Zeitspanne sollte für externe Achsen mit schwacher Leistung länger sein.

Syntax

```
IndInpos '('
  [MecUnit ':='] <variable (VAR) of mecunit>','
  [Axis ':='] <expression (IN) of num>')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Positioning during program execution*
Weitere unabhängige Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*
Prüfen des Geschwindigkeitsstatus für unabhängige Achsen	*IndSpeed - Unabhängiger Geschwindigkeitsstatus auf Seite 1361*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.97 IndSpeed - Unabhängiger Geschwindigkeitsstatus

Verwendung

`IndSpeed` wird verwendet, um zu prüfen, ob eine unabhängige Achse die ausgewählte Geschwindigkeit erreicht hat.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IndSpeed`.

Beispiel 1:

```
IndCMove Station_A, 2, 3.4;
WaitUntil IndSpeed(Station_A,2 \InSpeed) = TRUE;
WaitTime 0.2;
```

Warten, bis Achse `2` von `Station_A` die Geschwindigkeit 3,4 Grad/s erreicht hat.

Rückgabewert

Datentyp: `bool`

Die Tabelle beschreibt die Rückgabewerte von `IndSpeed \IndSpeed`:

Rückgabewert	Achsenstatus
TRUE	Hat die ausgewählte Geschwindigkeit erreicht.
FALSE	Hat die ausgewählte Geschwindigkeit nicht erreicht.

Die Tabelle beschreibt die Rückgabewerte von `IndSpeed \ZeroSpeed:`

Rückgabewert	Achsenstatus
TRUE	Null Geschwindigkeit.
FALSE	Geschwindigkeit nicht null

Argumente

```
IndSpeed ( MecUnit Axis [ \InSpeed ] | [ \ZeroSpeed ] )
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

`Axis`

Datentyp: `num`

Die Nummer der aktuellen Achse für die mechanische Einheit (1 bis 6).

`[ \InSpeed ]`

Datentyp: `switch`

`IndSpeed` gibt den Wert `TRUE` zurück, wenn die Achse die ausgewählte Geschwindigkeit erreicht hat; andernfalls wird `FALSE` zurückgegeben.

`[ \ZeroSpeed ]`

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.97 IndSpeed - Unabhängiger Geschwindigkeitsstatus

Independent Axis
Fortsetzung

`IndSpeed` gibt den Wert `TRUE` zurück, wenn die Achse null Geschwindigkeit hat; andernfalls wird `FALSE` zurückgegeben.

Wenn beide Argumente `\InSpeed` und `\ZeroSpeed` weggelassen werden, wird eine Fehlermeldung angezeigt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_AXIS_ACT`	Die Achse ist nicht aktiviert.
`ERR_AXIS_IND`	Die Achse befindet sich nicht im unabhängigen Modus.

Einschränkung

Die Funktion `IndSpeed\InSpeed` gibt in den folgenden Situationen immer den Wert `FALSE` zurück:

- Der Roboter befindet sich im Einrichtbetrieb mit reduzierter Geschwindigkeit.
- Die Geschwindigkeit wird über die Instruktion `VelSet` reduziert.
- Die Geschwindigkeit wird über das Produktionsfenster reduziert.

Eine Wartezeit von 0,2 Sekunden sollte nach der Instruktion hinzugefügt werden, um sicherzustellen, dass der korrekte Status erreicht wird. Diese Zeitspanne sollte für externe Achsen mit schwacher Leistung länger sein.

Syntax

```
IndSpeed '('
  [MecUnit ':='] <variable (VAR) of mecunit>','
  [Axis ':='] <expression (IN) of num>
  ['\' InSpeed] | ['\' ZeroSpeed]')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Unabhängige Achsen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - Positioning during program execution*
Weitere unabhängige Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*
Weitere Beispiele	*IndCMove - Unabhängige kontinuierliche Bewegung auf Seite 266*
Prüfen des Positionsstatus für unabhängige Achsen	*IndInpos - Unabhängige Achse in Positionsstatus auf Seite 1359*
Aktivieren von unabhängigen Gelenken	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Motion*, Typ *Arm*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.98 IOUnitState - Abrufen des aktuellen Status des E/A-Geräts

Verwendung

`IOUnitState` wird verwendet, um den aktuellen Status eines E/A-Geräts zu ermitteln. Der physische und logische Status definieren den Status eines E/A-Geräts.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `IOUnitState`.

Beispiel 1:

```
IF (IOUnitState("UNIT1" \Phys)=IOUNIT_PHYS_STATE_RUNNING) THEN
  ! Possible to access some signal on the I/O unit
ELSE
  ! Read/Write some signal on the I/O unit result in error
ENDIF
```

Test wird durchgeführt, um zu ermitteln, ob das E/A-Gerät `UNIT1` bereit ist und läuft.

Beispiel 2:

```
IF (IOUnitState("UNIT1" \Logic)=IOUNIT_LOG_STATE_DISABLED) THEN
  ! Unit is disabled by user from RAPID or FlexPendant
ELSE
  ! Unit is enabled.
ENDIF
```

Test wird durchgeführt, um zu ermitteln, ob das E/A-Gerät `UNIT1` deaktiviert ist.

Rückgabewert

Datentyp: `iounit_state`

Der Rückgabewert hängt davon ob, ob das optionale Argument `\Logic` oder `\Phys` oder kein optionales Argument verwendet wird.

Der logische Status des E/A-Geräts beschreibt den Status, den ein Benutzer für ein Gerät anweisen kann. In der folgenden Tabelle wird der Status eines E/A-Geräts bei Verwendung des optionalen Arguments `\Logic` beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
10	IOUNIT_LOG_STATE_DISABLED	Das E/A-Gerät wurde vom Benutzer über RAPID, FlexPendant oder Systemparameter deaktiviert.
11	IOUNIT_LOG_STATE_ENABLED	Das E/A-Gerät wurde vom Benutzer über RAPID, FlexPendant oder Systemparameter aktiviert. Standardeinstellung nach dem Start.

Wenn das Gerät vom Benutzer über den logischen Status aktiviert wurde und der Feldbustreiber versucht, ein E/A-Gerät in den physischen Status `IOUNIT_PHYS_STATE_RUNNING` zu versetzen, nimmt das E/A-Gerät aus verschiedenen Gründen möglicherweise andere Status an (siehe folgende Tabelle).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.98 IOUnitState - Abrufen des aktuellen Status des E/A-Geräts
RobotWare - OS
Fortsetzung

Der Status des E/A-Geräts bei Verwendung des optionalen Arguments `\Phys` wird in der folgenden Tabelle beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
20	IOUNIT_PHYS_STATE_DEACTIVATED	E/A-Gerät ist nicht in Betrieb (vom Benutzer deaktiviert)
21	IOUNIT_PHYS_STATE_RUNNING	E/A-Gerät ist in Betrieb
22	IOUNIT_PHYS_STATE_ERROR	E/A-Gerät läuft nicht wegen eines Laufzeitfehlers
23	IOUNIT_PHYS_STATE_UNCONNECTED	E/A-Gerät ist konfiguriert, aber nicht mit dem E/A-Netzwerk verbunden oder der E/A-Netzwerk wurde angehalten.
24	IOUNIT_PHYS_STATE_UNCONFIGURED	E/A-Gerät ist nicht konfiguriert, aber mit dem E/A-Netzwerk verbunden. [1]
25	IOUNIT_PHYS_STATE_STARTUP	E/A-Gerät befindet sich im Startmodus. [1]
26	IOUNIT_PHYS_STATE_INIT	E/A-Gerät wurde erstellt. [1]

Hinweis

Der Status des E/A-Geräts, wenn keines der optionalen Argumente `\Phys` oder `\Logic` verwendet wird, wird in der folgenden Tabelle beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
1	IOUNIT_RUNNING	E/A-Gerät ist in Betrieb
2	IOUNIT_RUNERROR	E/A-Gerät läuft nicht wegen eines Laufzeitfehlers
3	IOUNIT_DISABLE	E/A-Gerät wurde vom Anwender über RAPID oder FlexPendant deaktiviert
4	IOUNIT_OTHERERR	Andere Konfigurations- oder Startfehler

[1] Mit der aktuellen Version von Robotware - OS kann dieser Status nicht im RAPID-Programm abgerufen werden.

Argumente

```
IOUnitState (UnitName [\Phys] | [\Logic])
```

`UnitName`

Datentyp: `string`

Der Name des E/A-Geräts, das geprüft werden soll (mit dem konfigurierten Namen).

`[\Phys]`

Physical

Datentyp: `switch`

Bei Verwendung dieses Parameters wird der physische Status des E/A-Geräts gelesen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\Logic]`

Logical

Datentyp: `switch`

Bei Verwendung dieses Parameters wird der logische Status des E/A-Geräts gelesen.

Syntax

```
IOUnitState '('
  [ UnitName ':=' ] < expression (IN) of string >
  [ '\' Phys] | [ '\' Logic] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `iounit_state`.

Weitere Informationen

Information über	Siehe
Status des E/A-Geräts	*iounit_state - Status des E/A-Geräts auf Seite 1719*
Aktivieren eines E/A-Geräts	*IOEnable – Aktivieren eines E/A-Geräts auf Seite 294*
Deaktivieren eines E/A-Geräts	*IODisable – Deaktivieren eines E/A-Geräts auf Seite 291*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.99 IsBrakeCheckActive – Überprüfen Sie, ob die Bremsprüfung ausgeführt wird

Verwendung

`IsBrakeCheckActive` wird verwendet, um zu überprüfen, ob aktuell eine Bremsprüfung ausgeführt wird, d. h. ob die `CyclicBrakeCheck`- oder `BrakeCheck`-Verfahren auf einer der Abarbeitungsebenen aktiv sind (abarbeitend oder angehalten).

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `IsBrakeCheckActive`.

Beispiel 1:

```
WHILE IsBrakeCheckActive() = TRUE THEN
  WaitTime 1;
ENDWHILE
...
```

Überprüfen Sie, ob eine Bremsprüfungsroutine aktiv ist. Wenn Sie aktiv ist, warten Sie bis sie bereit ist.

Rückgabewert

Datentyp: `bool`

Die Funktion wird auf `TRUE` gestellt, wenn aktuell eine Bremsprüfung ausgeführt wird.

Syntax

```
IsBrakeCheckActive '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
`BrakeCheck`	*Bedienungsanleitung - IRC5 mit FlexPendant*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.100 IsCollFree - Überprüft die Position auf eine mögliche Kollision

Verwendung

`IsCollFree` wird verwendet, um zu Testen, ob der Roboter bei Positionierung an einer Achsenposition kollidieren würde, was als Argument für die Funktion bereitgestellt wird. In *MultiMove*-Systemen können mehrere Achsenpositionen für unterschiedliche Roboter angegeben werden.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `IsCollFree`.

Beispiel 1:

```
IF IsCollFree(testpos) THEN
  MoveAbsJ testpos, v50, fine, tool1;
ENDIF
```

Prüft für der Ausführung der Bewegung dahingehend, die Position (`jointtarget`) `testpos` eine kollisionsfreie Position ist.

Beispiel 2:

```
IF IsCollFree(testpos1 \Rob2Pos:=testpos2 \Rob3Pos:=testpos3) THEN
  TPWRITE "No collisions"
ENDIF
```

Angenommen, der aufrufende Task RAPID steuere Rob1. Prüft, ob die Positionen `testpos1` für Rob1, `testpos2` für Rob2 und `testpos3` für Rob3 in einer Kollision resultieren oder nicht.

Rückgabewert

Datentyp: `bool`

Die Funktion gibt `TRUE` zurück, wenn die angegebenen Achsenpositionen in keiner Kollision resultieren, andernfalls `FALSE`.

Argumente

```
IsCollFree ThisRobotPos [\Rob1Pos] [\Rob2Pos] [\Rob3Pos] [\Rob4Pos]
```

`ThisRobotPos`

Datentyp: `jointtarget`

Es wird geprüft, ob die Position kollisionsfrei ist oder nicht. Für den Roboter, der durch Aufruf von Task RAPID gesteuert wird.

`Rob1Pos`

Datentyp: `jointtarget`

Überprüfung auf eine Kollision, wenn auch Rob1 die Position `Rob1Pos` in *MultiMove*-Systemen hat. Kann nicht gesetzt werden, wenn der aufrufende Task RAPID Rob1, steuert, weil dann `ThisRobotPos` die Position dieses Roboters festlegt.

`Rob2Pos`

Datentyp: `jointtarget`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.100 IsCollFree - Überprüft die Position auf eine mögliche Kollision
RobotWare - OS
Fortsetzung

Überprüfung auf eine Kollision, wenn auch Rob2 die Position `Rob2Pos` in *MultiMove*-Systemen hat. Kann nicht gesetzt werden, wenn der aufrufende Task RAPID Rob2, steuert, weil dann `ThisRobotPos` die Position dieses Roboters festlegt.

`Rob3Pos`

Datentyp: `jointtarget`

Überprüfung auf eine Kollision, wenn auch Rob3 die Position `Rob3Pos` in *MultiMove*-Systemen hat. Kann nicht gesetzt werden, wenn der aufrufende Task RAPID Rob3, steuert, weil dann `ThisRobotPos` die Position dieses Roboters festlegt.

`Rob4Pos`

Datentyp: `jointtarget`

Überprüfung auf eine Kollision, wenn auch Rob4 die Position `Rob4Pos` in *MultiMove*-Systemen hat. Kann nicht gesetzt werden, wenn der aufrufende Task RAPID Rob4, steuert, weil dann `ThisRobotPos` die Position dieses Roboters festlegt.

Einschränkungen

- Wenn mit einem *MultiMove*-System kein optionale Argument geliefert wird, wird die Funktion die gemessene Position des Roboters verwenden. Um verlässliche Ergebnisse zu erhalten, sollten die in `IsCollFree` nicht bereitgestellten Roboter stillstehen.
- Überprüft nicht, ob die Bewegung zur angegebenen Position zu einer Kollision führen wird. Wie im vorstehenden Beispiel 1 kann die `testpos` kollisionsfrei sein, wobei die Bewegung in diese Position jedoch in einer Kollision resultieren kann.

Syntax

```
IsCollFree '('
  [ ThisRobotPos ':=' ] < expression (IN) of jointtarget > ','
  [ '\' Rob1Pos ':=' < expression (IN) of jointtarget >]
  [ '\' Rob2Pos ':=' < expression (IN) of jointtarget >]
  [ '\' Rob3Pos ':=' < expression (IN) of jointtarget >]
  [ '\' Rob4Pos ':=' < expression (IN) of jointtarget >]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Kollisionsprognose	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.101 IsCyclicBool - Prüft, ob eine persistente Variable eine Cyclic bool ist

Verwendung

`IsCyclicBool` wird verwendet, um zu testen, ob eine persistente boolesche Variable eine Cyclic bool ist oder nicht, das heißt, ob eine logische Bedingung durch die Instruktion `SetupCyclicBool` mit der persistenten booleschen Variable verbunden wurde.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `IsCyclicBool`.

Beispiel 1:

```
PERS bool cyclicflag1;

PROC main()
  TPWrite "cyclicflag1 is a cyclic bool:
       "\Bool:=IsCyclicBool(cyclicflag1);
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  TPWrite "cyclicflag1 is a cyclic bool:
       "\Bool:=IsCyclicBool(cyclicflag1);
  ...
```

Der Text `cyclicflag1 is a cyclic bool: FALSE` wird zuerst auf das FlexPendant geschrieben. Nach Ausführung von `SetupCyclicBool` wird die persistente boolesche Variable eine Cyclic bool und der zweite Text wird `cyclicflag1 is a cyclic bool: TRUE`.

Beispiel 2:

```
TASK PERS bool cyclicflag1;

PROC main()
  SetupCyclicBool cyclicflag1, di1=1 AND do2=1;
  TPWrite "cyclicflag1 is a cyclic bool: "
       \Bool:=IsCyclicBool("cyclicflag1");
  ...
```

Es wird eine Textzeichenfolge verwendet, um den zyklischen booleschen Namen in der `IsCyclicBool`-Funktion anzugeben. Der Text, der auf das FlexPendant geschrieben wird, ist `cyclicflag1 is a cyclic bool: TRUE`.

Beispiel 3

```
..
TPWrite "cyclicflag1 is a cyclic bool: "
     \Bool:=IsCyclicBool("cyclicflag1", \TaskName:="T_ROB1");
..
```

Es wird eine Textzeichenfolge verwendet, um den zyklischen booleschen Namen in die `IsCyclicBool`-Funktion einzugeben. Der Text, der auf das FlexPendant geschrieben wird, ist `cyclicflag1 is a cyclic bool: TRUE`, wenn `cyclicflag1` durch die Instruktion `SetupCyclicBool` in der RAPID-Task `T_ROB1` mit einer logischen Bedingung verbunden wurde, anderenfalls lautet der Text, der

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.101 IsCyclicBool - Prüft, ob eine persistente Variable eine Cyclic bool ist
RobotWare - OS
Fortsetzung

auf das FlexPendant geschrieben wird, `cyclicflag1 is a cyclic bool: FALSE`.

Rückgabewert

Datentyp: `bool`

Die Funktion gibt `TRUE` zurück, wenn eine logische Bedingung über die Instruktin `SetupCyclicBool` mit der persistenten booleschen Variable verbunden wurde, anderenfalls `FALSE`.

Argumente

```
IsCyclicBool (Flag | Name [\TaskRef] | [\TaskName])
```

`Flag`

Datentyp: `bool`

Die persistente boolesche Variable, die überprüft werden sollte.

`Name`

Datentyp: `string`

Der Name der persistenten booleschen Variablen, die überprüft werden sollte.

`[\TaskRef]`

Task Reference

Datentyp: `taskid`

Die Identität der Programmtask, in der die Instruktion `SetupCyclicBool` ausgeführt wurde. Dieses Argument sollte nur für eine Cyclic bool verwendet werden, die als `TASK PERS` deklariert wurde, und wenn die `IsCyclicBool`-Funktion aus einer RAPID-Task verwendet wird, welche die logische Bedingung nicht über die Instruktion `SetupCyclicBool` mit der persistenten booleschen Variablen verbunden hat.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“; für den Task `T_ROB1` lautet die Variablenidentität z. B. `T_ROB1Id`.

`[\TaskName]`

Datentyp: `string`

Der Name der Programmtask, in der die Instruktion `SetupCyclicBool` ausgeführt wurde. Dieses Argument sollte nur für eine Cyclic bool verwendet werden, die als `TASK PERS` deklariert wurde, und wenn die `IsCyclicBool`-Funktion aus einer RAPID-Task verwendet wird, welche die logische Bedingung nicht über die Instruktion `SetupCyclicBool` mit der persistenten booleschen Variablen verbunden hat.

Wenn keines der Argumente `\TaskRef` oder `\TaskName` angegeben ist, wird die aktuelle Task verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Die Namen der Cyclic bools werden als Zeichenfolge im System gespeichert. Für eine `PERS bool m1` lautet der gespeicherte Name `m1`. Für eine `TASK PERS bool m2` lautet der Name `"T_ROB1/m2"`, wenn die Konfiguration über die Instruktion `SetupCyclicBool` in der RAPID-Task `T_ROB1.` erfolgt ist.

Wenn die Funktion mit dem Argument `Flag` oder `Name` verwendet wird, wird zuerst geprüft, ob der persistente Name in der Liste derCyclic bools vorhanden ist, um festzustellen, ob es eine deklarierte PERS-Variable gibt, die über `SetupCyclicBool` mit einer Bedingung verbunden wurd. Wenn keine Cyclic bool mit diesem Namen gefunden wird, wird auch geprüft, ob es sich um eine `TASK PERS` handelt, indem die aktuelle ausgeführte Task vor den Namen des persistenten Namens gesetzt wird (`"T_ROB1/name"`).

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_TASKNAME`	Der Name der Programmtask im Argument `\TaskName` kann im System nicht gefunden werden.

Syntax

```
IsCyclicBool '('
  [ [ Flag ':=' ] <persistent (PERS) of bool>
  | [ [ Name ':=' ] <expression (IN) of string> ','
  ['\' TaskRef ':=' <variable (VAR) of taskid>]
  |['\' TaskName ':=' <expression (IN) of string>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Konfiguration einer Cyclic bool-Bedingung	*SetupCyclicBool - Konfiguration einer Cyclic bool-Bedingung auf Seite 722*
Entfernen einer Cyclic bool-Bedingung	*RemoveCyclicBool - Entfernen einer Cyclic bool-Bedingung auf Seite 601*
Alle Cyclic bool-Bedingungen entfernen	*RemoveAllCyclicBool - Alle Cyclic bool-Bedingungen entfernen auf Seite 599*
Zyklisch ausgewertete logische Bedingungen, *Cyclic bool*	*Anwendungshandbuch - Steuerungssoftware IRC5*
Konfigurieren von *Cyclic bool*	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.102 IsFile - Prüfen des Dateityps

Verwendung

Die Funktion `IsFile` bezieht Informationen über die angegebene Datei oder das angegebene Verzeichnis und prüft, ob die Datei bzw. das Verzeichnis vom angegebenen Typ ist. Wenn kein Typ angegeben ist, wird nur eine Existenzprüfung durchgeführt.

Das path-Argument gibt die Datei an. Lese-, Schreib- oder Abarbeitungsberechtigung ist für die angegebene Datei nicht erforderlich, aber alle im Pfadnamen aufgeführten Verzeichnisse, die zur Datei führen, müssen durchsucht werden können.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsFile`.

Siehe auch *Weitere Beispiele auf Seite 1374*.

Beispiel 1:

```
PROC printFT(string filename)
  IF IsFile(filename \Directory) THEN
    TPWrite filename+" is a directory";
    RETURN;
  ENDIF
  IF IsFile(filename \Fifo) THEN
    TPWrite filename+" is a fifo file";
    RETURN;
  ENDIF
  IF IsFile(filename \RegFile) THEN
    TPWrite filename+" is a regular file";
    RETURN;
  ENDIF
  IF IsFile(filename \BlockSpec) THEN
    TPWrite filename+" is a block special file";
    RETURN;
  ENDIF
  IF IsFile(filename \CharSpec) THEN
    TPWrite filename+" is a character special file";
    RETURN;
  ENDIF
ENDPROC
```

Dieses Beispiel gibt `filename` und Typ der angegebenen Datei am FlexPendant aus.

Rückgabewert

Datentyp: `bool`

Die Funktion gibt `TRUE` zurück, wenn der angegebene Typ und der tatsächliche Typ übereinstimmen. Andernfalls wird `FALSE` zurückgegeben. Wenn kein Typ angegeben ist und die Datei existiert, wird `TRUE` zurückgegeben, andernfalls `FALSE`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
IsFile (Path [\Directory] [\Fifo] [\RegFile] [\BlockSpec]
     [\CharSpec])
```

`Path`

Datentyp: `string`

Die angegebene Datei mit einem vollständigen oder relativen Pfad.

`[ \Directory ]`

Datentyp: `switch`

Datei ist ein Verzeichnis.

`[ \Fifo ]`

Datentyp: `switch`

Die Datei ist eine fifo-Datei.

`[ \RegFile ]`

Datentyp: `switch`

Die Datei ist eine normale Datei, d. h. eine normale Binär-, ISO 8859-1 (Latin-1)- oder UTF-8-Datei.

`[ \BlockSpec ]`

Datentyp: `switch`

Die Datei ist blockorientiert.

`[ \CharSpec ]`

Datentyp: `switch`

Die Datei ist zeichenorientiert.

Programmabarbeitung

Diese Funktion gibt einen Wert des Typs `bool` zurück, der angibt, ob eine Übereinstimmung vorliegt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Die Datei ist nicht vorhanden und es wurde ein Typ angegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.102 IsFile - Prüfen des Dateityps
RobotWare - OS
Fortsetzung

Weitere Beispiele

Weitere Beispiele für die Funktion `IsFile` sind unten aufgeführt.

Beispiel 1:

Dieses Beispiel implementiert einen generischen Durchlauf einer Verzeichnisstrukturfunktion.

```
PROC searchdir(string dirname, string actionproc)
  VAR dir directory;
  VAR string filename;
  IF IsFile(dirname \Directory) THEN
    OpenDir directory, dirname;
    WHILE ReadDir(directory, filename) DO
        ! .. and . is the parent and resp. this directory
        IF filename <> ".." AND filename <> "." THEN
          searchdir dirname+"/"+filename, actionproc;
        ENDIF
    ENDWHILE
    CloseDir directory;
  ELSE
    %actionproc% dirname;
  ENDIF
ERROR
  RAISE;
ENDPROC

PROC listfile(string filename)
  TPWrite filename;
ENDPROC

PROC main()
  ! Execute the listfile routine for all files found under the
  ! tree of HOME:
  searchdir "HOME:","listfile";
ENDPROC
```

Dieses Programm durchläuft die Verzeichnisstruktur unter "`HOME`:" und ruft für jede gefundene Datei die Prozedur `listfile` auf. Die Angabe `searchdir` ist der generische Teil, der weder den Beginn der Suche noch die aufzurufende Routine für jede Datei kennt. Sie verwendet `IsFile`, um zu prüfen, ob ein Unterverzeichnis oder eine Datei gefunden wurde, und verwendet den Mechanismus der späten Bindung, um für alle gefundenen Dateien die Prozedur aufzurufen, die in `actionproc` angegeben ist. Die Routine `actionproc` sollte eine Prozedur mit einem Parameter des Typs `string` sein.

Einschränkungen

Diese Funktion kann nicht für serielle Kanäle oder Feldbusse verwendet werden.

Bei der Verwendung für `FTP` oder bereitgestellte `NFS`-Datenträger werden die Datei- oder Typinformationen nicht immer aktualisiert. Vor der Verwendung von `IsFile` ist möglicherweise ein expliziter Befehl für den Suchpfad (mit der Instruktion `Open`) erforderlich, um ordnungsgemäße Informationen abzurufen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
Isfile '('
  [ Path ':=' ] < expression (IN) of string>
  [ '\' Directory ]
  | [ '\' Fifo ]
  | [ '\' RegFile ]
  | [ '\' BlockSpec ]
  | [ '\' CharSpec ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Verzeichnis	*dir – Dateiverzeichnisstruktur auf Seite 1689*
Öffnen eines Verzeichnisses	*OpenDir - Öffnen eines Verzeichnisses auf Seite 497*
Schließen eines Verzeichnisses	*CloseDir - Schließen eines Verzeichnisses auf Seite 153*
Lesen eines Verzeichnisses	*ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis auf Seite 1459*
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Kopieren einer Datei	*CopyFile - Kopieren einer Datei auf Seite 169*
Prüfen der Dateigröße	*FileSize - Abrufen der Größe einer Datei auf Seite 1304*
Prüfen der Dateisystemgröße	*FSSize - Abrufen der Größe eines Dateisystems auf Seite 1310*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.103 IsLeadThrough - Status der Durchführung prüfen

Verwendung

`IsLeadThrough` wird verwendet, um Informationen über den Status der Durchführung für einen TCP-Roboter zu erhalten.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `IsLeadThrough`.

Beispiel 1:

```
VAR bool leadthrough:=FALSE;
leadthrough:=IsLeadThrough();
```

Überprüft, ob in dieser Task eine Durchführung für den aktiven TCP-Roboter konfiguriert wurde. Wenn sie zum Beispiel in der RAPID-Task `T_ROB_L` ausgeführt wird, wird geprüft, ob die Durchführung für den TCP-Roboter `ROB_L` konfiguriert wurde.

Beispiel 2:

```
VAR bool leadthrough:=FALSE;
leadthrough:=IsLeadThrough(\MechUnit:=ROB_R);
```

Prüft, ob eine Durchführung für den TCP-Roboter `ROB_R` konfiguriert wurde.

Beispiel 3

```
VAR bool leadthrough:=FALSE;
leadthrough:=IsLeadThrough(\MechUnit:=ROB_R \Active);
```

Prüft, ob eine Durchführung für den TCP-Roboter `ROB_R` aktiv ist.

Beispiel 4:

```
SetLeadThrough \On \NoStopMove;
TPWrite "Set: "+ValToStr(IsLeadThrough(\MechUnit:=ROB_R \Set));
TPWrite "Active: "+ValToStr(IsLeadThrough(\MechUnit:=ROB_R
     \Active));
..
StopMove;
TPWrite "Set: "+ValToStr(IsLeadThrough(\MechUnit:=ROB_R \Set));
TPWrite "Active: "+ValToStr(IsLeadThrough(\MechUnit:=ROB_R
     \Active));
```

Prüft, ob eine Durchführung für den TCP-Roboter `ROB_R` konfiguriert wurde und aktiv ist. Die Durchführung ist erst aktiv, wenn eine `StopMove`-Instruktion abgearbeitet oder die Programmausführung angehalten wurde.

Die Druckausgabe lautet:

Set: TRUE

Active: FALSE

Set: TRUE

Active: TRUE

Rückgabewert

Datentyp: `bool`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Argumente

```
IsLeadThrough (\MechUnit \Active | \Set)
```

`\MechUnit`

Mechanical unit

Datentyp: `mecunit`

Der Name des TCP-Roboters.

Das Argument `\MechUnit` ist optional. Wenn es ausgelassen wird, erfolgt die Prüfung für die mechanische Einheit, die durch die vordefinierte RAPID-Variable `ROB_ID` dargestellt wird, die eine Referenz für den TCP-Roboter in der aktuellen Programmtask ist.

Wenn `\MechUnit` ausgelassen wird und `IsLeadThrough` von einer Nicht-Bewegungstask verwendet wird, erfolgt die Prüfung für den TCP-Roboter in den verbundenen Bewegungstask.

`\Active`

Datentyp: `switch`

`TRUE`**, wenn die Durchführung aktiviert wurde.**

`FALSE`**, wenn die Durchführung über `SetLeadThrough \Off` deaktiviert wurde.**

`FALSE` **kann auch zurückgegeben werden, wenn `SetLeadThrough \On \NoStopMove` ausgeführt wurde. Es ist der `StopMove`-Befehl oder ein Stopp der Programmausführung, der die Durchführung aktiviert.**

`\Set`

Datentyp: `switch`

`TRUE`**, wenn die Durchführung konfiguriert wurde.**

`FALSE`**, wenn die Durchführung zurückgesetzt wurde.**

Wenn keiner der Schalter verwendet wird, ist das Standardverhalten `\Set`.

Einschränkungen

Die mechanische Einheit muss ein TCP-Roboter sein.

Die Funktion `IsLeadThrough` kann nur für YuMi-Roboter verwendet werden.

Syntax

```
IsLeadThrough '('
  ['\'MecUnit ':=' < variable (VAR) of mecunit> ',']
  ['\'Active] | ['\'Set]')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Durchführung aktivieren und deaktivieren	*SetLeadThrough - Durchführung aktivieren und deaktivieren auf Seite 717*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.104 IsMechUnitActive - Prüfen, ob mechanische Einheit aktiv ist

Verwendung

`IsMechUnitActive` (*Is Mechanical Unit Active*) prüft, ob eine mechanische Einheit aktiviert ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsMechUnitActive`.

Beispiel 1:

```
IF IsMechUnitActive(SpotWeldGun)
  CloseGun SpotWeldGun;
```

Wenn die mechanische Einheit `SpotWeldGun` aktiv ist, wird die Routine `CloseGun` aufgerufen, in der die Zange (gun) geschlossen wird.

Rückgabewert

Datentyp: `bool`

Die Funktion gibt Folgendes zurück:

- `TRUE`, wenn die mechanische Einheit aktiv ist.
- `FALSE`, wenn die mechanische Einheit deaktiviert ist.

Argumente

```
IsMechUnitActive ( MechUnit )
```

`MechUnit`

Mechanical Unit

Datentyp: `mecunit`

Name der mechanischen Einheit

Syntax

```
IsMechUnitActive '('
  [ MechUnit ':='] < variable (VAR) of mecunit> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Aktivieren mechanischer Einheiten	*ActUnit - Aktivieren einer mechanischen Einheit auf Seite 30*
Deaktivieren mechanischer Einheiten	*DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Mechanische Einheiten	*mecunit – Mechanische Einheit auf Seite 1731*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.105 IsPers - Ist eine Persistente

Verwendung

`IsPers` wird verwendet, um zu prüfen, ob es sich bei einem Datenobjekt um eine persistente Variable (Persistente) handelt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsPers`.

Beispiel 1:

```
PROC procedure1 (INOUT num parameter1)
  IF IsVar(parameter1) THEN
    ! For this call reference to a variable
    ...
  ELSEIF IsPers(parameter1) THEN
    ! For this call reference to a persistent variable
    ...
  ELSE
    ! Should not happen
    EXIT;
  ENDIF
ENDPROC
```

Die Prozedur `procedure1` führt verschiedene Aktionen durch, je nachdem, ob der aktuelle Parameter `parameter1` eine Variable oder eine persistente Variable ist.

Rückgabewert

Datentyp: `bool`

TRUE, wenn der getestete aktuelle `INOUT`-Parameter eine persistente Variable ist. FALSE, wenn der getestete aktuelle `INOUT`-Parameter keine persistente Variable ist.

Argumente

```
IsPers (DatObj)
```

`DatObj`

Data Object

Datentyp: `anytype`

Der Name des formalen `INOUT`-Parameters.

Syntax

```
IsPers '('
  [ DatObj ':=' ] < var or pers (INOUT) of anytype > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.105 IsPers - Ist eine Persistente

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Testen, ob Variable	*IsVar - Prüfen, ob Datenobjekt eine Variable ist auf Seite 1388*
Parametertypen (Zugriffsmodi)	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Routines*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.106 IsStopMoveAct - Sind Flags zum Stoppen der Bewegung aktiv?

Verwendung

Mit `IsStopMoveAct`wird der Status der Flags zum Stoppen der Bewegung für die aktuelle oder verbundene Bewegungstask abgerufen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `IsStopMoveAct`.

Beispiel 1:

```
stopflag2:= IsStopMoveAct(\FromNonMoveTask);
```

`stopflag2` ist TRUE, wenn das Flag zum Stoppen der Bewegung aus Nicht-Bewegungstasks in der aktuellen oder verbundenen Bewegungstask gesetzt ist, andernfalls FALSE.

Beispiel 2:

```
IF IsStopMoveAct(\FromMoveTask) THEN
  StartMove;
ENDIF
```

Wenn das Flag zum Stoppen der Bewegung aus der Bewegungstask in der aktuellen Bewegungstask gesetzt ist, wird es mit der Instruktion `StartMove` zurückgesetzt.

Rückgabewert

Datentyp: `bool`

Der Rückgabewert ist TRUE, wenn das ausgewählte Flag zum Stoppen der Bewegung gesetzt ist, andernfalls ist der Rückgabewert FALSE.

Argumente

```
IsStopMoveAct ( [\FromMoveTask] | [\FromNonMoveTask] )
```

`[\FromMoveTask]`

Datentyp: `switch`

Mit `FromMoveTask` wird der Status des Flags zum Stoppen der Bewegung einer privaten Bewegungstask abgerufen.

Der Typ von Flag zum Stoppen der Bewegung kann nur gesetzt werden von:

- Der Bewegungstask selbst mit der Instruktion `StopMove`
- Nach Verlassen der `RestoPath`-Ebene im Programm
- Bei der Abarbeitung in einer asynchronen Fehlerbehandlungsroutine für Prozess- oder Bewegungsfehler vor jedem `StorePath` und nach jedem `RestoPath`

`[\FromNonMoveTask]`

Datentyp: `switch`

Mit `FromNonMoveTask` wird der Status des Flags zum Stoppen der Bewegung einer Nicht-Bewegungstask abgerufen. Dieser Typ von Flag zum Stoppen der Bewegung kann nur von Nicht-Bewegungstasks in einer verbundenen Task oder allen Bewegungstasks mit der Instruktion `StopMove` gesetzt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.106 IsStopMoveAct - Sind Flags zum Stoppen der Bewegung aktiv?

RobotWare - OS

Fortsetzung

Syntax

```
IsStopMoveAct '('
  ['\' FromMoveTask]
  | ['\' FromNonMoveTask] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Stoppen der Roboterbewegung	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Neustart der Roboterbewegung	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.107 IsStopStateEvent – Prüfen, ob Programmzeiger bewegt wurde

Verwendung

`IsStopStateEvent` gibt Informationen über die Bewegung des Programmzeigers (PZ) in der aktuellen Programmtask zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsStopStateEvent`.

Beispiel 1:

```
IF IsStopStateEvent (\PPMoved) = TRUE THEN
  ! PP has been moved during the last program stop
ELSE
  ! PP has not been moved during the last program stop
ENDIF

IF IsStopStateEvent (\PPToMain) THEN
  ! PP has been moved to main routine during the last program stop
ENDIF
```

Rückgabewert

Datentyp: `bool`

Status, ob und wie PZ beim letzten Programmstopp bewegt wurde.

`TRUE`, wenn PZ beim letzten Programmstopp bewegt wurde.

`FALSE`, wenn PZ beim letzten Programmstopp nicht bewegt wurde.

Wenn der PZ in die Hauptroutine bewegt wurde, gibt sowohl `\PPMoved` als auch `\PPToMain` den Wert `TRUE` zurück.

Wenn der PZ in eine Routine bewegt wurde, gibt sowohl `\PPMoved` als auch `\PPToMain` den Wert `TRUE` zurück.

Wenn der PZ in der Liste einer Routine verschoben wurde, gibt `\PPMovedTRUE` und `\PPToMainFALSE` zurück.

Nach dem Aufruf einer Serviceroutine (Abarbeitungskontext in Hauptprogrammsequenz belassen) gibt `\PPMove` den Wert `FALSE` und `\PPToMain` den Wert `FALSE` zurück.

Argumente

```
IsStopStateEvent ([\PPMoved] | [\PPToMain])
```

`[ \PPMoved ]`

Datentyp: `switch`

Prüfen, ob PP verschoben wurde.

`[ \PPToMain ]`

Datentyp: `switch`

Prüfen, ob PZ in "main" oder eine Routine verschoben wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.107 IsStopStateEvent – Prüfen, ob Programmzeiger bewegt wurde

RobotWare - OS

Fortsetzung

Einschränkungen

Diese Funktion kann in den meisten Fällen nicht in Vorwärts- oder Rückwärtsabarbeitung benutzt werden, da das System zwischen den einzelnen Schritten jeweils im Stoppzustand ist.

Syntax

```
IsStopStateEvent '('
  ['\' PPMoved] | ['\' PPToMain] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Erstellen von eigenen Instruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.108 IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus

Verwendung

`IsSyncMoveOn` wird verwendet, um zu testen, ob die aktuelle Programmtask des Typs `Motion Task` im Synchronbewegungsmodus ist.

Es ist auch möglich, in einer `Non Motion Task` zu testen, ob die verbundene `Motion Task` im Synchronbewegungsmodus ist. Die Systemparameter *Controller/Tasks/Use Mechanical Unit Group* definieren die verbundene `Motion Task`.

Wenn die `Motion Task` auf `StorePath`-Ebene abgearbeitet wird, prüft `IsSyncMoveOn`, ob die Task auf dieser Ebene im synchronisierten Modus ist, unabhängig vom synchronisierten Modus auf der ursprünglichen Ebene.

Die Instruktion `IsSyncMoveOn` wird normalerweise in einem *MultiMove*-System mit der Option *Coordinated Robots* verwendet, kann jedoch in jedem System und jeder Programmtask verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsSyncMoveOn`.

Beispiel 1:

Programmbeispiel in Task `T_ROB1`

```
PERS tasks task_list{2} := [ ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;

PROC main()
  ...
  MoveL p_zone, vmax, z50, tcp1;
  WaitSyncTask sync1, task_list;
  MoveL p_fine, v1000, fine, tcp1;
  syncmove;
  ...
ENDPROC

PROC syncmove()
  SyncMoveOn sync2, task_list;
  MoveL * \ID:=10, v100, z10, tcp1 \WOBJ:= rob2_obj;
  MoveL * \ID:=20, v100, fine, tcp1 \WOBJ:= rob2_obj;
  SyncMoveOff sync3;
  UNDO
    SyncMoveUndo;
ENDPROC
```

Programmbeispiel in Task `BCK1`

```
PROC main()
  ...
  IF IsSyncMoveOn() THEN
    ! Connected Motion Task is in synchronized movement mode
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.108 IsSyncMoveOn - Testen, ob im Synchronbewegungsmodus
RobotWare - OS
Fortsetzung

```
    ELSE
      ! Connected Motion Task is in independent mode
    ENDIF
    ...
  ENDPROC
```

Bei der Abarbeitung von `IsSyncMoveOn` in der Hintergrundtask `BCK1` wird getestet, ob die verbundene Bewegungstask gerade im Synchronbewegungsmodus ist.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn die aktuelle oder verbundene Programmtask im Synchronbewegungsmodus ist, andernfalls `FALSE`.

Programmabarbeitung

Test, ob die aktuelle oder verbundene Programmtask momentan im Synchronbewegungsmodus ist. Wenn die `MotionTask` auf `StorePath level` abgearbeitet wird, überprüft `SyncMoveOn`, ob die Task auf `StorePath level` und nicht der ursprünglichen Ebene in einer synchronisierten Bewegung ausgeführt wird.

Syntax

```
IsSyncMoveOn '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Identität für Synchronisierungspunkte	*syncident - Identität für Synchronisierungspunkte auf Seite 1814*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*
Festlegen von unabhängigen Bewegungen	*SyncMoveUndo - Festlegen von unabhängigen Bewegungen auf Seite 873*
Speichern der Bahn und Abarbeiten auf einer neuen Ebene	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.109 IsSysId - Testen der Systemidentität

Verwendung

`IsSysId` (*System Identity*) kann verwendet werden, um mithilfe der Seriennummer des Systems die Systemidentität zu überprüfen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsSysId`.

Beispiel 1:

```
IF NOT IsSysId("6400-1234") THEN
  ErrWrite "System identity fault", "Faulty system identity for
       this program";
  EXIT;
ENDIF
```

Das Programm ist für ein spezielles Robotersystem mit der Seriennummer `6400-1234` konzipiert und kann für kein anderes verwendet werden.

Rückgabewert

Datentyp: `bool`

TRUE = Die Seriennummer des Robotersystems ist dieselbe wie im Test angegeben.

FALSE = Die Seriennummer des Robotersystems ist nicht dieselbe wie im Test angegeben.

Argumente

```
IsSysId ( SystemId)
```

`SystemId`

Datentyp: `string`

die Seriennummer des Robotersystems, die die Systemidentität kennzeichnet.

Syntax

```
IsSysId '('
  [ SystemId ':=' ] < expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Lesen der Systeminformationen	*GetSysInfo - Abrufen von Informationen über das System auf Seite 1335*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.110 IsVar - Prüfen, ob Datenobjekt eine Variable ist

Verwendung

`IsVar` wird verwendet, um zu prüfen, ob es sich bei einem Datenobjekt um eine Variable handelt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `IsVar`.

Beispiel 1:

```
PROC procedure1 (INOUT num parameter1)
  IF IsVAR(parameter1) THEN
    ! For this call reference to a variable
    ...
  ELSEIF IsPers(parameter1) THEN
    ! For this call reference to a persistent variable
    ...
  ELSE
    ! Should not happen
    EXIT;
  ENDIF
ENDPROC
```

Die Prozedur `procedure1` führt verschiedene Aktionen durch, je nachdem, ob der aktuelle Parameter `parameter1` eine Variable oder eine Persistente ist.

Rückgabewert

Datentyp: `bool`

TRUE, wenn der getestete aktuelle INOUT-Parameter eine Variable ist. FALSE, wenn der getestete aktuelle INOUT-Parameter keine Variable ist.

Argumente

```
IsVar (DatObj)
```

`DatObj`

Data Object

Datentyp: `anytype`

Der Name des formalen `INOUT`-Parameters.

Syntax

```
IsVar '('
  [ DatObj ':=' ] < var or pers (INOUT) of anytype > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Testen, ob speichernd	*IsPers - Ist eine Persistente auf Seite 1379*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Parametertypen (Zugriffsmodi)	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Routines*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.111 Max - Gibt den größten von zwei Werten zurück.

Verwendung

`Max` Gibt das größte von zwei Argumenten zurück.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `Max`.

Beispiel 1:

```
reg1 := Max(reg2, reg3)
```

`reg1` wird dem größten Wert von `reg2` und `reg3` zugewiesen.

Rückgabewert

Datentyp: `num`

Gibt das größte von zwei Argumenten zurück.

Argumente

```
Max (A, B)
```

A

Datentyp: `num`

Erster numerischer Wert.

B

Datentyp: `num`

Zweiter numerischer Wert.

Syntax

```
Max '('
  [A ':='] < expression (IN) of num >','
  [B ':='] < expression (IN) of num >')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
`Min` Funktion	*Min - Gibt den kleinsten von zwei Werten zurück. auf Seite 1395*
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.112 MaxExtLinearSpeed - Maximale zusätzliche Achsengeschwindigkeit

Verwendung

`MaxExtLinearSpeed` (Maximale Zusatzachsen-Lineargeschwindigkeit) gibt die maximale Lineargeschwindigkeit für die zusätzlichen Achsen in der aktuellen Bewegungstask zurück.

Grundlegende Beispiele

Folgendes Beispiel veranschaulicht die Funktion `MaxExtLinearSpeed`.

Beispiel 1:

```
TPWrite "Max. Linear speed in mm/s for my axis="\Num:=
    MaxExtLinearSpeed ();
```

Die Nachricht `Max. Linear speed in mm/s for my axis = 5000` wird auf das FlexPendant geschrieben (Wert hängt von der Konfiguration ab).

Rückgabewert

Datentyp: `num`

Gibt die maximale (`vmax`) Lineargeschwindigkeit in mm/s für die zusätzlichen Achsen in dieser Task zurück.

Syntax

```
MaxExtLinearSpeed '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Definition von `vmax` Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Konfigurieren der maximalen Zusatzachsen-Lineargeschwindigkeit	*Technisches Referenzhandbuch - Systemparameter*, Parameter *Ext. Axis Linear Max Speed (m/s)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.113 MaxExtReorientSpeed - Maximale Zusatzachsen-Rotationsgeschwindigkeit

Verwendung

`MaxExtReorientSpeed` (Maximale Zusatzachsen-Rotationsgeschwindigkeit) gibt die maximale Rotationsgeschwindigkeit für die zusätzlichen Achsen in der aktuellen Bewegungstask zurück.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `MaxExtReorientSpeed`.

Beispiel 1:

```
TPWrite "Max. Rotational speed in deg/s for my axis="\Num:=
    MaxExtReorientSpeed ();
```

Die Nachricht `Max. Rotational speed in deg/s for my axis = 1000` wird auf das FlexPendant geschrieben (Wert hängt von der Konfiguration ab).

Rückgabewert

Datentyp: `num`

Gibt die maximale (`vmax`) Rotationsgeschwindigkeit in Grad/s für die zusätzlichen Achsen in dieser Task zurück.

Syntax

```
MaxExtReorientSpeed '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Definition von `vmax` Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Konfigurieren der maximalen Zusatzachsen-Rotationsgeschwindigkeit	*Technisches Referenzhandbuch - Systemparameter*, Parameter *Ext. Axis Rotational Max Speed (deg/s)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.114 MaxRobReorientSpeed - Maximale Umorientierungsgeschwindigkeit des Roboters

Verwendung

`MaxRobReorientSpeed` (Maximale Roboter-Umorientierungsgeschwindigkeit) gibt die maximale TCP-Umorientierungsgeschwindigkeit für den Roboter zurück.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `MaxRobReorientSpeed`.

Beispiel 1:

```
TPWrite "TCP Reorient Max Speed in deg/s for my robot="\Num:=
    MaxRobReorientSpeed ();
```

Die Nachricht `TCP Reorient Max Speed in deg/s for my robot = 500` wird auf das FlexPendant geschrieben (Wert hängt von der Konfiguration ab).

Rückgabewert

Datentyp: `num`

Gibt die maximale (`vmax`) TCP-Umorientieungsgeschwindigkeit in Grad für den verwendeten Roboter und übliche praktische TCP-Werte zurück.

Wenn extrem große TCP-Werte im Werkzeugrahmen verwendet werden, können Sie Ihre eigenen `speeddata` mit niedrigerer TCP-Umorientierungsgeschwindigkeit als von `MaxRobReorientSpeed` zurückgegeben erstellen.

Syntax

```
MaxRobReorientSpeed '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Definition von `vmax` Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Konfigurieren der max. TCP-Umorientierungsgeschwindigkeit	*Technisches Referenzhandbuch - Systemparameter*, Parameter *TCP Reorient Max Speed (deg/s)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.115 MaxRobSpeed - Höchstgeschwindigkeit des Roboters

Verwendung

`MaxRobSpeed` (*Maximum Robot Speed*) gibt die maximale TCP-Geschwindigkeit für den Robotertyp zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `MaxRobSpeed`.

Beispiel 1:

```
TPWrite "Max. TCP speed in mm/s for my robot="\Num:=MaxRobSpeed();
```

Die Nachricht `Max. TCP speed in mm/s for my robot = 7000` wird auf das FlexPendant geschrieben (Wert hängt von der Konfiguration ab).

Rückgabewert

Datentyp: `num`

Gibt die maximale (`vmax`) TCP-Geschwindigkeit in mm/s für das verwendete Robotermodell und übliche praktische TCP-Werte zurück.

Wenn extrem große TCP-Werte im Werkzeugrahmen verwendet werden, können Sie Ihre eigenen `speeddata` mit größerer TCP-Geschwindigkeit als von `MaxRobSpeed` zurückgegeben erstellen und `VelSet` verwenden, um eine größere Geschwindigkeit zuzulassen.

Syntax

```
MaxRobSpeed '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Definition von `vmax` Geschwindigkeit	*speeddata - Geschwindigkeitsdaten auf Seite 1795*
Definition der Höchstgeschwindigkeit	*VelSet - Ändern der programmierten Geschwindigkeit auf Seite 1038*
Konfigurieren der maximalen TCP-Geschwindigkeit	*Technisches Referenzhandbuch - Systemparameter*, Parameter *TCP Linear Max Speed (m/s)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.116 Min - Gibt den kleinsten von zwei Werten zurück.

Verwendung

`Min` Gibt das kleinste von zwei Argumenten zurück.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `Min`.

Beispiel 1:

```
reg1 := Min(reg2, reg3)
```

`reg1` wird dem kleinsten Wert von `reg2` und `reg3` zugewiesen.

Rückgabewert

Datentyp: `num`

Gibt das kleinste von zwei Argumenten zurück.

Argumente

```
Min (A, B)
```

A

Datentyp: `num`

Erster numerischer Wert.

B

Datentyp: `num`

Zweiter numerischer Wert.

Syntax

```
Min '('
  [A ':='] < expression (IN) of num >','
  [B ':='] < expression (IN) of num >')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
`Max` Funktion	*Max - Gibt den größten von zwei Werten zurück. auf Seite 1390*
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.117 MirPos - Spiegeln einer Position

Verwendung

`MirPos` (*Mirror Position*) wird verwendet, um die Translations- und Rotationsteile einer Position zu spiegeln.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `MirPos`.

Beispiel 1:

```
CONST robtarget p1:= [...];
VAR robtarget p2;
PERS wobjdata mirror:= [...];
...
p2 := MirPos(p1, mirror);
```

`p1` ist ein Roboterziel, das eine Position des Roboters und eine Orientierung des Werkzeugs speichert. Diese Position wird in der xy-Ebene des Koordinatensystems gespiegelt, das durch `mirror` relativ zum Welt-Koordinatensystem definiert wurde. Das Ergebnis sind neue robtarget-Daten, die in `p2` gespeichert werden.

Rückgabewert

Datentyp: `robtarget`

Die neue Position, die der gespiegelten Position der eingegebenen Position entspricht.

Argumente

```
MirPos (Point MirPlane [\WObj] [\MirY])
```

`Point`

Datentyp: `robtarget`

Die eingegebene Roboterposition. Der Orientierungsteil dieser Position definiert die aktuelle Orientierung des Werkzeug-Koordinatensystems.

`MirPlane`

Mirror Plane

Datentyp: `wobjdata`

Die Werkobjektdaten, die die Spiegelebene definieren. Die Spiegelebene ist die xy-Ebene des Objekt-Koordinatensystems, die in `MirPlane` definiert ist. Das Objekt-Koordinatensystem wird relativ zum Anwender-Koordinatensystem (ebenfalls in `MirPlane`) definiert und dieses wiederum relativ zum Welt-Koordinatensystem.

`[\WObj]`

Work Object

Datentyp: `wobjdata`

Die Werkobjektdaten, die das Objekt- und das Anwender-Koordinatensystem definieren, in Relation zu dem die eingegebene Position *Point* definiert wird. Wenn

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

dieses Argument weggelassen wird, erfolgt die Definition relativ zum Welt-Koordinatensystem.

HINWEIS!

Wenn die Position mit einem aktiven Werkobjekt erstellt wird, muss im Argument auf dieses Werkobjekt verwiesen werden.

`[\MirY]`

Mirror Y

Datentyp: `switch`

Wenn dieser Schalter ausgelassen wird, was dem Standardverhalten entspricht, wird das Werkzeug-Koordinatensystem mit Bezug auf die x- und die z-Achse gespiegelt. Wenn dieser Schalter angegeben wird, wird das Werkzeug-Koordinatensystem mit Bezug auf die y- und die z-Achse gespiegelt.

Einschränkungen

Die Roboterkonfiguration wird als Teil der eingegebenen Roboterzieldaten nicht neu berechnet.

Bei Verwendung eines Koordinatensystems muss sich die koordinierte Einheit in derselben Task befinden wie der Roboter.

Syntax

```
MirPos '('
  [ Point ':=' ] < expression (IN) of robtarget> ','
  [ MirPlane ':='] <expression (IN) of wobjdata> ','
  ['\' WObj ':=' <expression (IN) of wobjdata> ]
  ['\' MirY ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `robtarget`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*
Positionsdaten	*robtarget - Positionsdaten auf Seite 1778*
Werkobjektdaten	*wobjdata - Werkobjektdaten auf Seite 1852*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.118 MOD - berechnet einen Ganzzahlmodulo

Verwendung

`MOD` ist eine bedingte Anweisung, um den Modulo zu berechnen, d. h. den Rest nach einer Ganzzahldivision.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `MOD`.

Beispiel 1:

```
reg1 := 14 MOD 4;
```

Der Rückgabewert ist `2`, weil `14` geteilt durch `4` den Modulo `2` ergibt.

Beispiel 2:

```
VAR dnum mydnum1 := 11;
VAR dnum mydnum2 := 5;
VAR dnum mydnum3;
...
mydnum3 := mydnum1 MOD mydnum2;
```

Der Rückgabewert ist `1`, weil `11` geteilt durch `5` den Modulo `2` ergibt.

Rückgabewert

Datentyp: `num`, `dnum`

Gibt den Modulo zurück, d. h. den Rest nach einer Ganzzahldivision.

Syntax

```
<expression of num> MOD <expression of num>
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

```
<expression of dnum> MOD <expression of dnum>
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
num - Numerische Werte	*num - Numerische Werte auf Seite 1740*
dnum – Doppelte numerische Werte	*dnum – Doppelte numerische Werte auf Seite 1690*
DIV	*DIV - berechnet eine Ganzzahldivision auf Seite 1288*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.119 ModExist - Überprüfen, ob ein Programmmodul vorhanden ist.

Verwendung

`ModExist` (*Module Exist*) wird verwendet, um zu überprüfen, ob ein angegebenes Modul in der Programmtask vorhanden ist.

Die Suche erfolgt zunächst für geladene Module, und wenn kein geladenes Modul gefunden wurde, werden installierte Module gesucht.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ModExist`.

Beispiel 1:

```
VAR bool mod_exist;
mod_exist:=ModExist ("MyModule");
```

Wenn das Modul `MyModule` im Task vorhanden ist, gibt die Funktion `TRUE` zurück. Wenn nicht, gibt die Funktion `FALSE` zurück.

Rückgabewert

Datentyp: `bool`

TRUE, wenn das Modul gefunden wurde, andernfalls FALSE.

Argumente

```
ModExist (ModuleName)
```

`ModuleName`

Datentyp: `string`

Name des zu suchenden Moduls.

Syntax

```
ModExist '('
  [ ModuleName ':=' ] < expression (IN) of string > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Suchen des Änderungszeitpunkts für geladenes Modul	*ModTimeDnum - Abrufen des Dateiänderungszeitpunkts für das geladene Modul auf Seite 1400*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.120 ModTimeDnum - Abrufen des Dateiänderungszeitpunkts für das geladene Modul

Verwendung

`ModTimeDnum` (*Modify Time*) wird verwendet, um den Zeitpunkt der letzten Dateiänderung des geladenen Moduls abzurufen. Das Modul wird mit seinem Namen angegeben und muss sich im Taskspeicher befinden. Die Zeit wird in Sekunden seit 00:00:00 GMT, 1. Jan. 1970 gemessen. Die Zeit wird mit dem Datentyp `dnum` und optional auch mit dem Datentyp `stringdig` zurückgegeben.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `ModTimeDnum`.

Siehe auch *Weitere Beispiele auf Seite 1401*.

Beispiel 1:

```
MODULE mymod
  VAR dnum mytime;
  PROC printMyTime()
    mytime := ModTimeDnum("mymod");
    TPWrite "My time is "+ValToStr(mytime);
  ENDPROC
ENDMODULE
```

Rückgabewert

Datentyp: `dnum`

Die Zeit in Sekunden seit 00:00:00 GMT, 1. Jan. 1970.

Argumente

```
ModTimeDnum ( Object [\StrDig] )
```

`Object`

Datentyp: `string`

Name des Moduls.

`[\StrDig]`

String Digit

Datentyp: `stringdig`

Um die Modulladezeit in einer `stringdig`-Darstellung abzurufen.

Programmabarbeitung

Diese Funktion gibt einen numerischen Wert zurück, der den letzten Änderungszeitpunkt der Datei angibt, bevor sie als Programmmodul in das System geladen wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_MOD_NOT_LOADED	In der Programmtask ist kein Modul mit dem angegebenen Namen vorhanden.

Weitere Beispiele

Weitere Beispiele für die Funktion ModTimeDnum sind unten aufgeführt.

Beispiel 1:

```
IF FileTimeDnum ("HOME:/mymod.mod" \ModifyTime) > ModTimeDnum
     ("mymod") THEN
  UnLoad "HOME:/mymod.mod";
  Load \Dynamic, "HOME:/mymod.mod";
ENDIF
```

Dieses Programm lädt ein Modul erneut, wenn eine neuere Version der Quelldatei vorhanden ist. Es ruft mit ModTimeDnum den letzten Ladezeitpunkt des angegebenen Moduls ab und vergleicht ihn mit der FileTimeDnum ("HOME:/mymod.mod" \ModifyTime) an der Quelle. Wenn das Modul an der Quelle neuer ist, entfernt das Programm das Modul und lädt es erneut.

Einschränkungen

Diese Funktion gibt immer 0 zurück, wenn sie in einem Modul verwendet wird, das verschlüsselt ist oder als gemeinsam genutztes Modul installiert wurde.

Syntax

```
ModTimeDnum '('
  [Object ':='] <expression (IN) of string>
  ['\' StrDig ':=' <variable (VAR) of stringdig>]')'
```

Eine Funktion mit einem Rückgabewert des Typs dnum.

Weitere Informationen

Information über	Siehe
Abrufen von Zeitinformationen über eine Datei	*FileTimeDnum - Abrufen von Zeitinformationen über eine Datei auf Seite 1307*
Zeichenfolge, die nur Ziffern enthält	*stringdig - Zeichenfolge, die nur Ziffern enthält auf Seite 1808*
Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten	*StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten auf Seite 1518*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.121 MotionPlannerNo - Abrufen der verbundenen Bewegungsplanungsnummer

Verwendung

`MotionPlannerNo` gibt die Nummer der verbundenen Bewegungsplanung zurück. Wenn die Instruktion `MotionPlannerNo` in einem Bewegungstask abgearbeitet wird, gibt sie ihre Planungsnummer zurück. Wenn die Instruktion `MotionPlannerNo` nicht in einem Bewegungstask abgearbeitet wird, gibt sie die verbundene Bewegungsplanungsnummer gemäß den Einstellungen in den Systemparametern zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `MotionPlannerNo`.

Beispiel 1:

```
!Motion task T_ROB1
PERS string buffer{6} := ["", "", "", "", "", ""];
VAR num motion_planner;

PROC main()
  ...
  MoveL point, v1000, fine, tcp1;
  motion_planner := MotionPlannerNo();
  buffer{motion_planner} := "READY";
  ...
ENDPROC

!Background task BCK1
PERS string buffer{6};
VAR num motion_planner;
VAR string status;

PROC main()
  ...
  motion_planner := MotionPlannerNo();
  status := buffer{motion_planner};
  ...
ENDPROC

!Motion T_ROB2
PERS string buffer{6};
VAR num motion_planner;

PROC main()
  ...
  MoveL point, v1000, fine, tcp1;
  motion_planner := MotionPlannerNo();
  buffer{motion_planner} := "READY";
  ...
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.121 MotionPlannerNo - Abrufen der verbundenen Bewegungsplanungsnummer

RobotWare - OS

Fortsetzung

```
!Background task BCK2
PERS string buffer{6};
VAR num motion_planner;
VAR string status;

PROC main()
  ...
  motion_planner := MotionPlannerNo();
  status := buffer{motion_planner};
  ...
ENDPROC
```

Verwenden Sie die Funktion `MotionPlannerNo`, um zu ermitteln, welche Bewegungsplanungsnummer mit der Task verbunden ist. Exakt der gleiche Code kann in alle Bewegungstasks und Hintergrundtasks implementiert werden. Jede Hintergrundtask kann dann den Status ihrer verbundenen Bewegungstask prüfen.

Rückgabewert

Datentyp:`num`

Die Nummer der verbundenen Bewegungsplanung. Für Nicht-Bewegungstask wird die Bewegungsplanungsnummer der zugehörigen mechanischen Einheit zurückgegeben.

Der Rückgabewert liegt im Bereich 1 ... 6.

Syntax

```
MotionPlannerNo '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*Technisches Referenzhandbuch - Systemparameter*, Abschnitt *Controller - Task*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.122 NonMotionMode - Lesen des Nichtbewegungs-Abarbeitungsmodus

Verwendung

`NonMotionMode` (*Non-Motion Execution Mode*) wird verwendet, um den aktuellen Nichtbewegungs-Abarbeitungsmodus der Programmtask zu lesen. Der Nichtbewegungs-Abarbeitungsmodus wird am FlexPendant unter dem Menüeintrag *ABB\Bedienfeld\Überwachung* aktiviert bzw. deaktiviert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `NonMotionMode`.

Beispiel 1:

```
IF NonMotionMode() =TRUE THEN
  ...
ENDIF
```

Der Programmabschnitt wird nur abgearbeitet, wenn sich der Roboter im Nichtbewegungs-Abarbeitungsmodus befindet.

Rückgabewert

Datentyp: `bool`

Der aktuelle Nichtbewegungs-Modus wie in nachstehender Tabelle definiert.

Rückgabewert	Symbolische Konstante	Kommentar
0	FALSE	Die Nichtbewegungs-Abarbeitung ist nicht aktiv.
1	TRUE	Die Nichtbewegungs-Abarbeitung ist aktiv.

Argumente

```
NonMotionMode ( [ \Main] )
```

`[ \Main ]`

Datentyp: `switch`

Gibt den aktuellen Abarbeitungsmodus für die verbundene Bewegungstask zurück. Wird in Multitasking-Systemen zum Abrufen des aktuellen Abarbeitungsmodus für die aktuelle Task verwendet, falls es sich um eine Bewegungstask oder eine verbundene Bewegungstask handelt und die Funktion `NonMotionMode` in einer Nicht-Bewegungstask abgearbeitet wird.

Wenn dieses Argument weggelassen wird, spiegelt der Rückgabewert stets den aktuellen Abarbeitungsmodus für die Programmtask, die die Funktion `NonMotionMode` abarbeitet.

Beachten Sie, dass der Abarbeitungsmodus mit dem System und nicht mit einer Task verbunden ist. Das bedeutet, dass alle Tasks in einem System für die Funktion `NonMotionMode` denselben Wert zurückgeben.

Syntax

```
NonMotionMode '(' ['\' Main] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Lesen des Betriebsmodus	*OpMode - Lesen des Betriebsmodus auf Seite 1414*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.123 NOT – Kehrt einen logischen Wert um

Verwendung

`NOT` ist eine bedingte Anweisung, die verwendet wird, um einen logischen Wert (true/false) umzukehren.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die bedingte Anweisung `NOT`.

Beispiel 1:

```
VAR bool mybool;
mybool := NOT mybool;
```

Wenn `mybool` gleich `TRUE` ist, dann ist der Rückgabewert FALSE.

Wenn `mybool` gleich `FALSE` ist, dann ist der Rückgabewert TRUE.

Beispiel 2:

```
VAR bool a;
VAR bool b;
VAR bool c;
...
c := a AND (NOT b);
```

Der Rückgabewert `c` ist `TRUE`, wenn `a` gleich `TRUE` ist und `b` gleich `FALSE` ist.

Rückgabewert

Datentyp: `bool`

Gibt den umgekehrten Wert wieder.

Syntax

```
NOT <logical term>
```

Weitere Informationen

Information über	Siehe
AND	*AND – Bewertet einen logischen Wert auf Seite 1175*
OR	*OR – Bewertet einen logischen Wert auf Seite 1415*
XOR	*XOR – Bewertet einen logischen Wert auf Seite 1634*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.124 NOrient - Normalisieren der Orientierung

Verwendung

`NOrient` (*Normalize Orientation*) wird zum Normalisieren einer nicht normalisierten Orientierung (quaternion) verwendet.

Beschreibung

Eine Orientierungen muss normalisiert werden, d. h. die Summe der Quadrate muss gleich 1 sein.

$$q_1^2+q_2^2+q_3^2+q_4^2=1$$

xx0500002452

Wenn die Orientierung geringfügig unnormalisiert ist, kann sie normalisiert werden. Der Normalisierungsfehler ist der Absolutwert der Summe der Quadrate von den Orientierungskomponenten. Die Orientierung wird als geringfügig unnormalisiert betrachtet, wenn der Normalisierungsfehler größer als 0,00001 und kleiner als 0,1 ist. Wenn der Normalisierungsfehler größer als 0,1 ist, ist die Orientierung unbrauchbar.

$$ABS(\sqrt{q_1^2+q_2^2+q_3^2+q_4^2}-1)=normerr$$

xx0500002453

normerr > 0.1 nicht verwendbar

normerr > 0.00001 AND normerr <= 0.1 geringfügig nicht normalisiert

normerr <= 0.00001 normalisiert

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `NOrient`.

Beispiel 1:

Eine geringfügig unnormalisierte Position (0,707170, 0, 0, 0,707170).

$$ABS(\sqrt{0{,}707170^2+0^2+0^2+0{,}707170^2}-1) = 0{,}0000894$$

$$0{,}0000894 > 0{,}00001 \Rightarrow \text{unnormalized}$$

xx0500002451

```
VAR orient unnormorient := [0.707170, 0, 0, 0.707170];
VAR orient normorient;
...
...
normorient := NOrient(unnormorient);
```

Die Normalisierung der Orientierung (0,707170, 0, 0, 0,707170) wird (0,707107, 0, 0, 0,707107).

Rückgabewert

Datentyp: `orient`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.124 NOrient - Normalisieren der Orientierung

RobotWare - OS

Fortsetzung

Die normalisierte Orientierung.

Argumente

```
NOrient (Rotation)
```

Rotation

Datentyp: `orient`

Die zu normalisierende Orientierung.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_ORIENT_VALUE`	Falscher Ausrichtungswert in der `NOrient`-Funktion.

Syntax

```
NOrient '('
  [Rotation ':='] <expression (IN) of orient> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `orient`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.125 NumToDnum – Konvertieren von num in dnum

Verwendung

`NumToDnum` konvertiert ein `num` in ein `dnum`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `NumToDnum`.

Beispiel 1:

```
VAR num mynum:=55;
VAR dnum mydnum:=0;
mydnum:=NumToDnum(mynum);
```

Der num-Wert 55 wird über die Funktion als `dnum`-Wert 55 zurückgegeben.

Rückgabewert

Datentyp: `dnum`

Der Rückgabewert des Typs `dnum` kann denselben Wert haben wie der Eingangswert des Typs `num`.

Argumente

```
NumToDnum (Value)
```

`Value`

Datentyp: `num`

Der zu konvertierende numerische Wert.

Syntax

```
NumToDnum '('
  [ Value ':=' ] < expression (IN) of num > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
`num`-Datentyp	*num - Numerische Werte auf Seite 1740*
`Dnum`-Datentyp	*dnum – Doppelte numerische Werte auf Seite 1690*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.126 NumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge

Verwendung

`NumToStr` (*Numeric To String*) Numerischen Wert in eine Zeichenfolge konvertieren

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `NumToStr`.

Beispiel 1:

```
VAR string str;
str := NumToStr(0.38521,3);
```

Die Variable `str` erhält den Wert `"0.385"`.

Beispiel 2:

```
reg1 := 0.38521;
str := NumToStr(reg1, 2\Exp);
```

Die Variable `str` erhält den Wert `"3.85E-01"`.

Rückgabewert

Datentyp: `string`

Der in eine Zeichenfolge konvertierte numerische Wert mit der angegebenen Anzahl an Dezimalstellen und mit Exponent, falls angefordert. Der numerische Wert wird bei Bedarf gerundet. Das Dezimalkomma wird unterdrückt, wenn keine Nachkommastellen vorhanden sind.

Argumente

```
NumToStr (Val Dec [\Exp])
```

`Val`

Value

Datentyp: `num`

Der zu konvertierende numerische Wert.

`Dec`

Decimals

Datentyp: `num`

Anzahl an Dezimalstellen. Die Anzahl an Dezimalstellen darf nicht als negativer Wert oder mit größerer Genauigkeit als die verfügbare Genauigkeit für numerische Werte angegeben werden.

`[\Exp]`

Exponent

Datentyp: `switch`

Verwenden eines Exponenten im Rückgabewert.

Syntax

```
NumToStr '('
  [ Val ':=' ] <expression (IN) of num>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
[ Dec ':=' ] <expression (IN) of num>
['\' Exp ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - String functions*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Basic elements*
Numerischen Wert in eine Zeichenfolge konvertieren	*DnumToStr - Konvertieren eines numerischen Werts in eine Zeichenfolge auf Seite 1291*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.127 Offs - Verschieben einer Roboterposition

Verwendung

`Offs` wird verwendet, um einer Roboterposition im Objekt-Koordinatensystem einen Offset hinzuzufügen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `Offs`.

Siehe auch *Weitere Beispiele auf Seite 1412*.

Beispiel 1:

```
MoveL Offs(p2, 0, 0, 10), v1000, z50, tool1;
```

Der Roboter wird an einen Punkt `10` mm von der Position `p2` (in z-Richtung) bewegt.

Beispiel 2:

```
p1 := Offs (p1, 5, 10, 15);
```

Die Roboterposition `p1` wird um `5` mm in x-Richtung, `10` mm in y-Richtung und `15` mm in z-Richtung verschoben.

Rückgabewert

Datentyp: `robtarget`

Die verschobenen Positionsdaten.

Argumente

```
Offs (Point XOffset YOffset ZOffset)
```

`Point`

Datentyp: `robtarget`

Die zu verschiebenden Positionsdaten.

`XOffset`

Datentyp: `num`

Die Verschiebung in die x-Richtung des Objekt-Koordinatensystems.

`YOffset`

Datentyp: `num`

Die Verschiebung in die y-Richtung des Objekt-Koordinatensystems.

`ZOffset`

Datentyp: `num`

Die Verschiebung in die z-Richtung des Objekt-Koordinatensystems.

Weitere Beispiele

Weitere Beispiele für die Funktion `Offs` sind unten aufgeführt.

Beispiel 1:

```
PROC pallet (num row, num column, num distance, PERS tooldata tool,
     PERS wobjdata wobj)
```

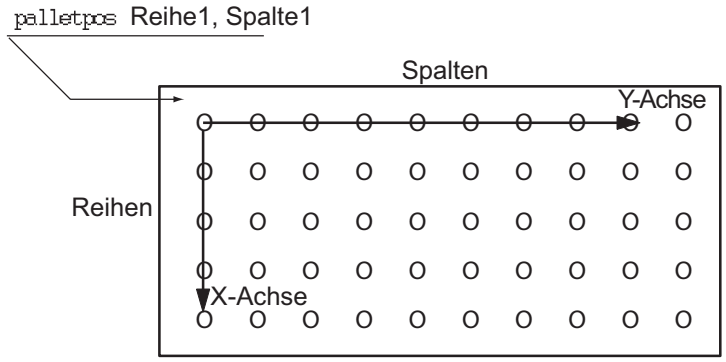
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
VAR robtarget palletpos:=[[0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0],
     [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]];
palettpos := Offs (palettpos, (row-1)*distance, (column-1)*distance,
     0);
MoveL palettpos, v100, fine, tool\WObj:=wobj;
ENDPROC
```

Eine Routine für das Aufnehmen von Teilen von einer Palette wird erstellt. Jede Palette wird als ein Werkobjekt definiert (siehe Abbildung unten). Das aufzunehmende Teil (Zeile und Spalte) und der Abstand zwischen den Teilen werden als Eingangsparameter angegeben. Das Inkrementieren des Zeilen- und Spaltenindex erfolgt außerhalb der Routine.

Die Abbildung zeigt die Position und die Orientierung der Palette, die durch Definition eines Werkobjekts angegeben werden.



xx050002300_

Syntax

```
Offs '('
  [Point ':='] <expression (IN) of robtarget> ','
  [XOffset ':='] <expression (IN) of num> ','
  [YOffset ':='] <expression (IN) of num> ','
  [ZOffset ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `robtarget`.

Weitere Informationen

Information über	Siehe
Positionsdaten	*robtarget - Positionsdaten auf Seite 1778*
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Mathematics*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.128 OpMode - Lesen des Betriebsmodus

Verwendung

`OpMode`(*Operating Mode*) wird verwendet, um den aktuellen Betriebsmodus des Systems zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `OpMode`.

Beispiel 1:

```
TEST OpMode()
CASE OP_AUTO:
  ...
CASE OP_MAN_PROG:
  ...
CASE OP_MAN_TEST:
  ...
DEFAULT:
  ...
ENDTEST
```

Verschiedene Programmabschnitte werden abhängig vom aktuellen Betriebsmodus abgearbeitet.

Rückgabewert

Datentyp: `symnum`

Der aktuelle Betriebsmodus wie in nachstehender Tabelle definiert.

Rückgabewert	Symbolische Konstante	Kommentar
0	OP_UNDEF	Undefinierter Betriebsmodus
1	OP_AUTO	Automatischer Betriebsmodus
2	OP_MAN_PROG	Einrichtbetrieb max. 250 mm/s
3	OP_MAN_TEST	Einrichtbetrieb, volle Geschwindigkeit 100 %

Syntax

```
OpMode '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `symnum`.

Weitere Informationen

Information über	Siehe
Verschiedene Betriebsmodi	*Bedienungsanleitung - IRC5 mit FlexPendant*
Lesen des Abarbeitungsmodus	*RunMode - Lesen des Abarbeitungsmodus auf Seite 1488*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.129 OR – Bewertet einen logischen Wert

Verwendung

OR ist ein bedingter Ausdruck, der verwendet wird, um einen logischen Wert (true/false) zu berechnen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion OR.

Beispiel 1:

```
VAR num a;
VAR num b;
VAR bool c;
...
c := a>5 OR b=3;
```

Der Rückgabewert von c ist TRUE, wenn a größer als 5 oder b gleich 3 ist. Ansonsten ist der Rückgabewert FALSE.

Beispiel 2:

```
VAR num mynum;
VAR string mystring;
VAR bool mybool;
VAR bool result;
...
result := mystring="Hello" OR mynum<15 AND mybool;
```

Der Rückgabewert von result ist TRUE, wenn mystring gleich "Hello" ist. Oder wenn beide mynum kleiner als 15 sind und mybool gleich TRUE ist. Ansonsten ist der Rückgabewert FALSE.

Die AND-Anweisung wird zuerst bewertet, danach die OR-Anweisung. Dies wird durch die Klammern in der Zeile unten angezeigt.

```
result := mystring="Hello" OR (mynum<15 AND mybool);
```

Rückgabewert

Datentyp: bool

Der Rückgabewert ist TRUE, wenn einer oder beide der bedingten Ausdrücke richtig sind; ansonsten ist der Rückgabewert FALSE.

Syntax

```
<expression of bool> OR <expression of bool>
```

Eine Funktion mit einem Rückgabewert des Typs bool.

Weitere Informationen

Information über	Siehe
AND	*AND – Bewertet einen logischen Wert auf Seite 1175*
XOR	*XOR – Bewertet einen logischen Wert auf Seite 1634*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.129 OR – Bewertet einen logischen Wert

RobotWare - OS

Fortsetzung

Information über	Siehe
NOT	*NOT – Kehrt einen logischen Wert um auf Seite 1406*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.130 OrientZYX - Erzeugt eine Orientierung aus Eulerschen Winkeln.

Verwendung

`OrientZYX` (*Orient from Euler ZYX angles*) wird verwendet, um eine Variable des Typs "orient" aus Eulerschen Winkeln zu erzeugen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `OrientZYX`.

Beispiel 1:

```
VAR num anglex;
VAR num angley;
VAR num anglez;
VAR pose object;
...
object.rot := OrientZYX(anglez, angley, anglex)
```

Rückgabewert

Datentyp: `orient`

Die aus Eulerschen Winkeln erzeugte Orientierung.

Argumente

```
OrientZYX (ZAngle YAngle XAngle)
```

Hinweis

Die Rotationen erfolgen in der nachstehenden Reihenfolge:

1. Rotation um die z-Achse
2. Rotation um die neue y-Achse
3. Rotation um die neue x-Achse

`ZAngle`

Datentyp: `num`

Die Rotation in Grad um die Z-Achse.

`YAngle`

Datentyp: `num`

Die Rotation in Grad um die Y-Achse.

`XAngle`

Datentyp: `num`

Die Rotation in Grad um die X-Achse.

Syntax

```
OrientZYX '('
  [ZAngle ':='] <expression (IN) of num> ','
  [YAngle ':='] <expression (IN) of num> ','
  [XAngle ':='] <expression (IN) of num> ')'
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.130 OrientZYX - Erzeugt eine Orientierung aus Eulerschen Winkeln.

RobotWare - OS

Fortsetzung

Eine Funktion mit einem Rückgabewert des Typs `orient`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Mathematics*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.131 ORobT - Entfernen einer Programmverschiebung von einer Position

Verwendung

`ORobT` (*Object Robot Target*) wird verwendet, um eine Roboterposition aus dem Programmverschiebungs-Koordinatensystem in das Objekt-Koordinatensystem zu transformieren und/oder einen Offset für die externen Achsen zu entfernen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ORobT`.

Siehe auch *Weitere Beispiele auf Seite 1420*.

Beispiel 1:

```
VAR robtarget p10;
VAR robtarget p11;
VAR num wobj_diameter;

p10 := CRobT(\Tool:=tool1 \WObj:=wobj_diameter);
p11 := ORobT(p10);
```

Die aktuellen Positionen der Roboterachse und der externen Achsen werden in `p10` und `p11` gespeichert. Die Werte in `p10` beziehen sich auf das Koordinatensystem ProgDisp/ExtOffs. Die Werte in `p11` beziehen sich auf das Objektkoordinatensystem ohne Programmverschiebung und Versatz an den externen Achsen.

Rückgabewert

Datentyp: `robtarget`

Die umgewandelten Positionsdaten.

Argumente

```
ORobT (OrgPoint [\InPDisp] | [\InEOffs])
```

`OrgPoint`

Original Point

Datentyp: `robtarget`

Der zu transformierende Ursprungspunkt.

`[\InPDisp]`

In Program Displacement

Datentyp: `switch`

Gibt die TCP-Position im `ProgDisp`-Koordinatensystem wieder, d. h. sie entfernt nur den Versatz der externen Achsen.

`[\InEOffs]`

In External Offset

Datentyp: `switch`

Gibt die externen Achsen im Versatz-Koordinatensystem zurück, d. h. entfernt nur die Programmverschiebung für den Roboter.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.131 ORobT - Entfernen einer Programmverschiebung von einer Position

RobotWare - OS

Fortsetzung

Weitere Beispiele

Weitere Beispiele für die Verwendung der Funktion `ORobT` sind unten aufgeführt.

Beispiel 1:

```
p10 := ORobT(p10 \InEOffs );
```

Die Funktion `ORobT` entfernt nur eine aktive Programmverschiebung und belässt die TCP-Position in Relation zum Objekt-Koordinatensystem. Die externen Achsen bleiben im Offset-Koordinatensystem.

Beispiel 2:

```
p10 := ORobT(p10 \InPDisp );
```

Die Funktion `ORobT` entfernt alle Offsets der externen Achsen. Die TCP-Position bleibt im `ProgDisp`-Koordinatensystem.

Syntax

```
ORobT '('
  [ OrgPoint ':=' ] < expression (IN) of robtarget>
  ['\' InPDisp] | ['\' InEOffs] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `robtarget`.

Weitere Informationen

Information über	Siehe
Definition einer Programmverschiebung für den Roboter	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537* *PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem auf Seite 542*
Definition des Offsets für externe Achsen	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195* *EOffsSet - Aktivieren eines Offsets für zusätzliche Achsen durch bekannte Werte auf Seite 198*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - Coordinate systems*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.132 ParIdPosValid - Gültige Roboterposition für Parameteridentifikation

Verwendung

`ParIdPosValid` (*Parameter Identification Position Valid*) prüft, ob die Roboterposition für die aktuelle Parameteridentifikation gültig ist, z. B. für die Lastidentifikation eines Werkzeugs oder einer Nutzlast.

Diese Instruktion kann nur in der Task `main` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ParIdPosValid`.

Beispiel 1:

```
VAR jointtarget joints;
VAR bool valid_joints{12};

! Check if valid robot type
IF ParIdRobValid(TOOL_LOAD_ID) <> ROB_LOAD_VAL THEN
  EXIT;
ENDIF
! Read the current joint angles
joints := CJointT();
! Check if valid robot position
IF ParIdPosValid (TOOL_LOAD_ID, joints, valid_joints) = TRUE THEN
  ! Valid position for load identification
  ! Continue with LoadId
  ...
ELSE
  ! Not valid position for one or several axes for load
  ! identification
  ! Move the robot to the output data given in variable joints
  ! and do ParIdPosValid once again
  ...
ENDIF
```

Prüfen, ob die Roboterposition gültig ist, bevor die Lastidentifikation des Werkzeugs erfolgt.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn die Roboterposition für die aktuelle Parameteridentifikation gültig ist.

`FALSE`, wenn die Roboterposition für die aktuelle Parameteridentifikation nicht gültig ist.

Argumente

```
ParIdPosValid (ParIdType Pos AxValid [\ConfAngle])
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.132 ParIdPosValid - Gültige Roboterposition für Parameteridentifikation

RobotWare - OS

Fortsetzung

`ParIdType`

Datentyp: `paridnum`

Typ der Parameteridentifikation wie in nachstehender Tabelle definiert.

Wert	Symbolische Konstante	Kommentar
1	`TOOL_LOAD_ID`	Identifizieren der Werkzeuglast
2	`PAY_LOAD_ID`	Identifizieren der Nutzlast (Ref.-Instruktion GripLoad)
3	`IRBP_K`	Identifizieren der Last des externen Manipulators IRBP K
4	`IRBP_L`	Identifizieren der Last des externen Manipulators IRBP L
4	`IRBP_C`	Identifizieren der Last des externen Manipulators IRBP C
4	`IRBP_C_INDEX`	Identifizieren der Last des externen Manipulators IRBP C_INDEX
4	`IRBP_T`	Identifizieren der Last des externen Manipulators IRBP T
5	`IRBP_R`	Identifizieren der Last des externen Manipulators IRBP R
6	`IRBP_A`	Identifizieren der Last des externen Manipulators IRBP A
6	`IRBP_B`	Identifizieren der Last des externen Manipulators IRBP B
6	`IRBP_D`	Identifizieren der Last des externen Manipulators IRBP D

`Pos`

Datentyp: `jointtarget`

Die Variable gibt die tatsächlichen Achsenwinkel für alle Roboter und externen Achsen an. Die Variable wird gemäß der nachstehenden Tabelle durch `ParIdPosValid` aktualisiert.

Eingangsachswert	Ausgangsachswert
Gültig	Nicht geändert
Nicht gültig	In geeigneten Wert geändert

`AxValid`

Datentyp: `bool`

Datenfeldvariable mit 12 Elementen, die 6 Roboter- und 6 externen Achsen entsprechen. Die Variable wird gemäß der nachstehenden Tabelle durch `ParIdPosValid` aktualisiert.

Eingangsachswert in Pos	Ausgangsstatus in "AxValid"
Gültig	TRUE
Nicht gültig	FALSE

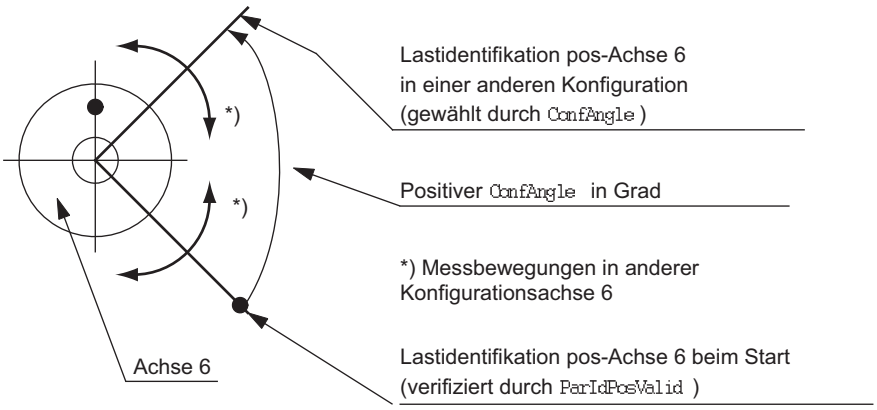
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[ \ConfAngle ]`

Datentyp: `num`

Optionales Argument für die Angabe eines bestimmtem Konfigurationswinkels +/- Grad, der zur Parameteridentifikation verwendet werden soll.



xx0500002493

Standard +90 Grad, wenn dieses Argument angegeben ist.

Min. + oder - 30 Grad. Optimum + oder - 90 Grad.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_PID_RAISE_PP`	Es tritt ein Fehler auf.

Syntax

```
ParIdPosValid '('
  [ ParIdType ':=' ] <expression (IN) of paridnum> ','
  [ Pos ':=' ] <variable (VAR) of jointtarget> ','
  [ AxValid ':=' ] <array variable {*} (VAR) of bool>
  [ '\' ConfAngle ':=' <expression (IN) of num> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Parameteridentifikationstyp	*paridnum - Parameteridentifikationstyp auf Seite 1749*
Gültiger Robotertyp	*ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation auf Seite 1424*
Lastidentifikation der Werkzeug- oder Nutzlast	*LoadId - Lastidentifikation der Werkzeug- oder Nutzlast auf Seite 351*
Lastidentifikation von Positioniergeräten (IRBP)	*ManLoadIdProc - Lastidentifikation des Manipulators IRBP auf Seite 359*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.133 ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation

Verwendung

`ParIdRobValid` (*Parameter Identification Robot Valid*) prüft, ob der Roboter- oder Manipulatortyp für die aktuelle Parameteridentifikation gültig ist, z. B. für die Lastidentifikation eines Werkzeugs oder einer Nutzlast.

Diese Instruktion kann nur in der Haupttask `T_ROB1` oder in einem *MultiMove*-System in Bewegungstasks verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ParIdRobValid`.

Beispiel 1:

```
TEST ParIdRobValid (TOOL_LOAD_ID)
  CASE ROB_LOAD_VAL:
    ! Possible to do load identification of tool in actual robot
         type
    ...
  CASE ROB_LM1_LOAD_VAL:
    ! Only possible to do load identification of tool with
    ! IRB 6400FHD if actual load < 200 kg
    ...
  CASE ROB_NOT_LOAD_VAL:
    ! Not possible to do load identification of tool in actual
         robot type
    ...
ENDTEST
```

Rückgabewert

Datentyp: `paridvalidnum`

Prüfen, ob die angegebene Parameteridentifikation mit dem aktuellen Roboter- oder Manipulatortyp wie in nachstehender Tabelle definiert ausgeführt werden kann.

Wert	Symbolische Konstante	Kommentar
10	ROB_LOAD_VAL	Gültiger Roboter- oder Manipulatortyp für die aktuelle Parameteridentifikation
11	ROB_NOT_LOAD_VAL	Nicht gültiger Typ für die aktuelle Parameteridentifikation
12	ROB_LM1_LOAD_VAL	Gültiger Robotertyp IRB 6400FHD für die aktuelle Parameteridentifikation, wenn aktuelle Last < 200 kg

Argumente

```
ParIdRobValid(ParIdType [\MechUnit] [\AxisNo])
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`ParIdType`

Datentyp: `paridnum`

Typ der Parameteridentifikation wie in nachstehender Tabelle definiert.

Wert	Symbolische Konstante	Kommentar
1	TOOL_LOAD_ID	Identifizieren der Roboter-Werkzeuglast
2	PAY_LOAD_ID	Identifizieren der Roboter-Nutzlast (Ref.-Instruktion GripLoad)
3	IRBP_K	Identifizieren der Last des externen Manipulators IRBP K
4	IRBP_L	Identifizieren der Last des externen Manipulators IRBP L
4	IRBP_C	Identifizieren der Last des externen Manipulators IRBP C
4	IRBP_C_INDEX	Identifizieren der Last des externen Manipulators IRBP C_INDEX
4	IRBP_T	Identifizieren der Last des externen Manipulators IRBP T
5	IRBP_R	Identifizieren der Last des externen Manipulators IRBP R
6	IRBP_A	Identifizieren der Last des externen Manipulators IRBP A
6	IRBP_B	Identifizieren der Last des externen Manipulators IRBP B
6	IRBP_D	Identifizieren der Last des externen Manipulators IRBP D

`[ \MechUnit ]`

Mechanical Unit

Datentyp: `mecunit`

Mechanische Einheit, die zur Lastidentifizierung benutzt wird. Darf nur für externen Manipulator angegeben werden. Wenn dieses Argument weggelassen wird, wird der TCP-Roboter in der Task verwendet.

`[ \AxisNo ]`

Axis number

Datentyp: `num`

Die Achsennummer in der mechanischen Einheit, die die zu identifizierende Last hält. Darf nur für externen Manipulator angegeben werden.

Wenn das Argument `\MechUnit` benutzt wird, muss auch `\AxisNo` verwendet werden. Das Argument `\AxisNo` kann nicht ohne `\MechUnit` benutzt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.133 ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_PID_RAISE_PP	Es tritt ein Fehler auf.

Syntax

```
ParIdRobValid '('
  [ParIdType ':='] <expression (IN) of paridnum>
  ['\' MechUnit ':=' <variable (VAR) of mecunit>]
  ['\' AxisNo ':=' <expression (IN) of num>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs paridvalidnum.

Weitere Informationen

Information über	Siehe
Parameteridentifikationstyp	*paridnum - Parameteridentifikationstyp auf Seite 1749*
Zu identifizierende mechanische Einheit	*mecunit – Mechanische Einheit auf Seite 1731*
Ergebnis dieser Funktion	*paridvalidnum - Ergebnis von ParIdRobValid auf Seite 1751*
Gültige Roboterposition	*ParIdPosValid - Gültige Roboterposition für Parameteridentifikation auf Seite 1421*
Lastidentifikation der Roboter-Werkzeuglast oder -Nutzlast	*LoadId - Lastidentifikation der Werkzeug- oder Nutzlast auf Seite 351*
Lastidentifikation von Positioniergerätlasten	*ManLoadIdProc - Lastidentifikation des Manipulators IRBP auf Seite 359*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.134 PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus

Verwendung

`PathLengthGet` wird verwendet, um den aktuellen Wert des Zählers zu lesen, der die vom TCP des Roboters zurückgelegte Weglänge misst. Der Rückgabewert ist in Millimetern und wird immer relativ zum Werkobjekt gemessen.

Diese Funktion kann jederzeit aufgerufen werden, es ist jedoch ratsam, sie aufzurufen, wenn der Roboter stillsteht, um ein vorhersehbares Verhalten zu erhalten.

Der Bahnlängenwert wird für den TCP-Roboter in der aktuellen oder verbundenen Bewegungsaufgabe gelesen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `PathLengthGet`.

Beispiel 1:

```
PathLengthStart;
MoveJ p10, v1000, z50, L10tip;
...
MoveL p40, v1000, fine, L10tip;
PathLengthStop;
TPWrite "PathLengthGet: "+ValToStr(PathLengthGet());
PathLengthReset;
```

In diesem Beispiel wird der Wert des Zählers ausgelesen, der die vom TCP des Roboters zurückgelegte Bahnlänge misst. Der Wert wird dann in den FlexPendant geschrieben.

Programmabarbeitung

Die Bahnlängenmessung gilt jeweils für die nächste abgearbeitete Roboterbewegungsinstruktion und ist so lange wirksam, bis eine `PathLengthStop`-Instruktion ausgeführt wird.

Die Bahnlängenmessung wird ausgeschaltet und der Bahnlängenmessungszähler wird auf null gesetzt, wenn ein Befehl `PathLengthReset` ausgeführt wird. Der Standardwert, Bahnlängenmessung aus, wird automatisch eingestellt:

- bei Verwendung des Neustart-Modus *RAPID zurücksetzen*.
- beim Laden eines neuen Programms oder eines neuen Moduls.
- und bei Beginn der Programmabarbeitung vom Anfang.
- beim Bewegen des Programmzeigers zu Hauptroutine.
- beim Bewegen des Programmzeiger zu einer Routine.
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

Einschränkungen

Bahnlängenmessung gelten nur für TCP Roboter.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.134 PathLengthGet - Liest den aktuellen Bahnlängenwert des Zählers aus

RobotWare - OS

Fortsetzung

Syntax

```
PathLengthGet '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
`PathLengthReset`	*PathLengthReset - Setzt den aktuellen Bahnlängenwert des Zählers zurück auf Seite 511*
`PathLengthStart`	*PathLengthStart - Aktiviert den Zähler, der die Bahnlänge überwacht auf Seite 513*
`PathLengthStop`	*PathLengthStop - Stoppt den Zähler, der die Bahnlänge überwacht auf Seite 515*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.135 PathLevel - Abrufen der aktuellen Bahnebene

Verwendung

`PathLevel` wird verwendet, um die aktuelle Bahnebene abzurufen. Daher zeigt diese Funktion, ob die Task auf der ersten ursprünglichen Ebene abgearbeitet wird oder ob die ursprüngliche Bewegungsbahn gespeichert und eine neue temporäre Bewegung abgearbeitet wird. Weitere Informationen über *Path Recovery* finden Sie unter *Anwendungshandbuch - Steuerungssoftware IRC5*.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PathLevel`.

Siehe auch *Weitere Beispiele auf Seite 1429*.

Beispiel 1:

```
VAR num level;
level:= PathLevel();
```

Die Variable `level` ist 1, wenn sie in einer ursprünglichen Bewegungsbahn abgearbeitet wird, oder 2, wenn sie in einer temporären neuen Bewegungsbahn abgearbeitet wird.

Rückgabewert

Datentyp: `num`

Es gibt zwei mögliche Rückgabewerte.

Rückgabewert	Beschreibung
1	Abarbeitung in der ursprünglichen Bewegungsbahn.
2	Abarbeitung in der Ebene "StorePath", einer temporären neuen Bewegungsbahn.

Weitere Beispiele

Ein weiteres Beispiel für die Verwendung der Funktion `PathLevel` ist unten aufgeführt.

Beispiel 1:

```
...
MoveL p100, v100, z10, tool1;
StopMove;
StorePath;
p:= CRobT(\Tool:=tool1);
!New temporary movement
MoveL p1, v100, fine, tool1;
...
level:= PathLevel();
...
MoveL p, v100, fine, tool1;
RestoPath;
StartMove;
...
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.135 PathLevel - Abrufen der aktuellen Bahnebene
RobotWare - OS
Fortsetzung

Die Variable `level` ist 2.

Einschränkungen

Die RobotWare-Option Path Recovery muss installiert sein, damit die Funktion `PathLevel` auf Bahnebene 2 verwendbar ist.

Syntax

```
PathLevel '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Path Recovery	*Anwendungshandbuch - Steuerungssoftware IRC5*
Pfad speichern und wiederherstellen	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846* *RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Bewegung stoppen und starten	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806* *StopMove - Stoppen der Roboterbewegung auf Seite 839*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.136 PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde

Verwendung

Mit `PathRecValidBwd` wird überprüft, ob die Bahnaufzeichnung aktiviert ist und ob eine aufgezeichnete Rückwärtsbahn verfügbar ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PathRecValidBwd`.

Siehe auch *Weitere Beispiele auf Seite 1432*.

Beispiel 1:

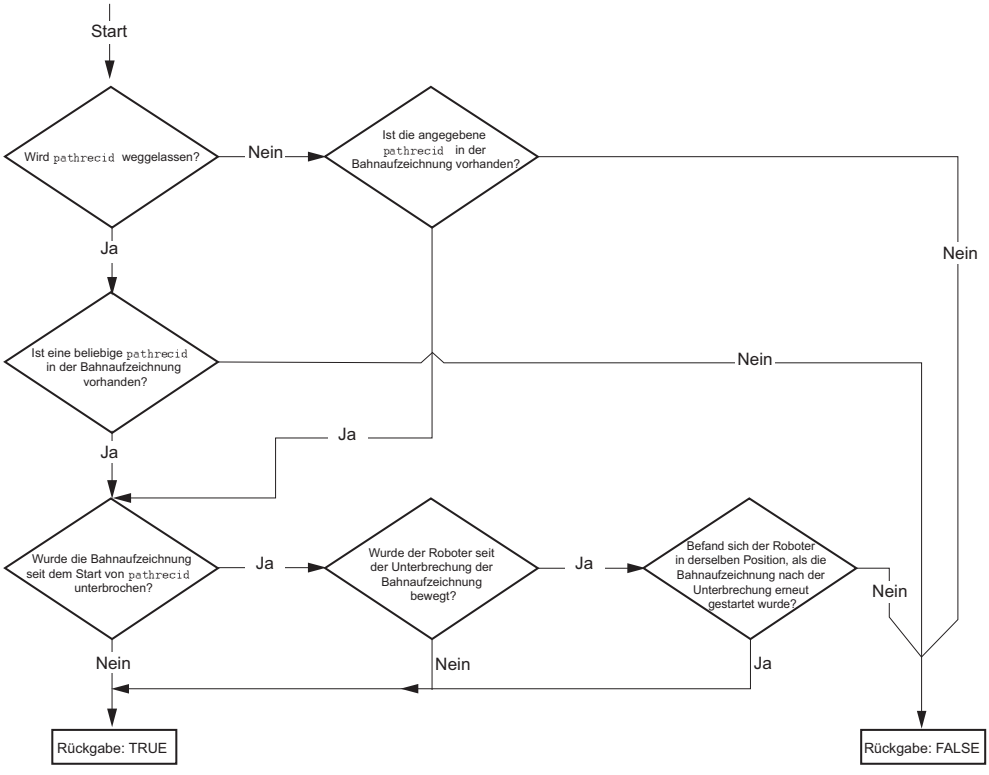
```
VAR bool bwd_path;
VAR pathrecid fixture_id;
bwd_path := PathRecValidBwd (\ID:=fixture_id);
```

Die Variable `bwd_path` wird auf TRUE gesetzt, wenn eine Rückwärtsbewegung auf der Bahn bis zur Position mit dem Bezeichner `fixture_id` möglich ist. Wenn nicht, wird `bwd_path` auf FALSE gesetzt.

Rückgabewert

Datentyp: `bool`

Der Rückgabewert der Funktion kann im folgenden Flussdiagramm bestimmt werden:



xx0500002132

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.136 PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde

Path Recovery
Fortsetzung

Argumente

```
PathRecValidBwd ([\ID])
```

`[\ID]`

Identifier

Datentyp: `pathrecid`

Variable, die den Namen der Aufzeichnungsstartposition angibt. Der Datentyp `pathrecid` weist keinen Wert auf (Non-Value) und wird nur als Bezeichner für die Benennung der Aufzeichnungsposition verwendet.

Programmabarbeitung

Bevor die Bahnaufzeichnung mit `PathRecMoveBwd` zur Rückwärtsbewegung angewiesen wird, kann mit `PathRecValidBwd` geprüft werden, ob eine gültige aufgezeichnete Bahn vorhanden ist.

Weitere Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `PathRecValidBwd`.

Beispiel 1:

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
bwd_path := PathRecValidBwd (\ID := id1);
```

Die Bahnaufzeichnung wird gestartet und zwei Bewegungsinstruktionen werden abgearbeitet. `PathRecValidBwd` gibt TRUE zurück und die verfügbare Rückwärtsbahn ist:

Von `p2` bis `p1` zur Startposition.

Beispiel 2:

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1;
PathRecStop \Clear;
bwd_path:= PathRecValidBwd (\ID := id1);
```

Die Bahnaufzeichnung wird gestartet und zwei Bewegungsinstruktionen werden abgearbeitet. Dann wird die Bahnaufzeichnung gestoppt und gelöscht. `PathRecValidBwd` gibt FALSE zurück.

Beispiel 3

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
PathRecStart id2;
MoveL p2, vmax, z50, tool1;
bwd_path := PathRecValidBwd ();
```

Die Bahnaufzeichnung wird gestartet und eine Bewegungsinstruktion wird abgearbeitet. Dann wird ein zusätzlicher Bahnbezeichner gefolgt von einer Bewegungsinstruktion angegeben. `PathRecValidBwd` gibt TRUE zurück und die Rückwärtsbahn ist:

Von `p2` bis `p1`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.136 PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde

Path Recovery

Fortsetzung

Beispiel 4:

```
PathRecStart id1;
MoveL p1, vmax, z50, tool1;
WaitSyncTask sync101, tasklist_r1o1;
MoveL p2, vmax, z50, tool1;
bwd_path1 := PathRecValidBwd ();
bwd_path2 := PathRecValidBwd (\ID := id1);
```

Die Abarbeitung des oben dargestellten Programms führt dazu, dass der booleschen Variablen `bwd_path1` TRUE zugewiesen wird, weil eine gültige Rückwärtsbahn zu der `WaitSyncTask`-Anweisung vorhanden ist. Der booleschen Variablen `bwd_path2` wird FALSE zugewiesen, weil es möglich ist, sich hinter eine `WaitSyncTask`-Anweisung zurückzubewegen.

Syntax

```
PathRecValidBwd '('
  ['\' ID ':=' < variable (VAR) of pathrecid >] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Bezeichner der Bahnaufzeichnung	*pathrecid - Bezeichner der Bahnaufzeichnung auf Seite 1753*
Starten/Stoppen der Bahnaufzeichnung	*PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *PathRecStop - Stoppen der Bahnaufzeichnung auf Seite 531*
Abspielen der Bahnaufzeichnung rückwärts	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*
Prüfen, ob eine gültige Vorwärtsbahn existiert	*PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde auf Seite 1434*
Abspielen der Bahnaufzeichnung vorwärts	*PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen auf Seite 525*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.137 PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde

Verwendung

Mit `PathRecValidFwd` wird überprüft, ob die Bahnaufzeichnung für die Vorwärtsbewegung verwendet werden kann. Damit eine Vorwärtsbewegung mit der Bahnaufzeichnung möglich ist, muss die Bahnaufzeichnung zuvor angewiesen worden sein, eine Rückwärtsbewegung auszuführen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PathRecValidFwd`.

Siehe auch *Weitere Beispiele auf Seite 1435*.

Beispiel 1:

```
VAR bool fwd_path;
VAR pathrecid fixture_id;

fwd_path:= PathRecValidFwd (\ID:=fixture_id);
```

Die Variable `fwd_path` wird auf `TRUE` gesetzt, wenn eine Vorwärtsbewegung bis zur Position mit dem Bezeichner `fixture_id` möglich ist. Wenn nicht, wird `fwd_path` auf `FALSE` gesetzt.

Rückgabewert

Datentyp: `bool`

Der Rückgabewert von `PathRecValidFwd`**ohne** angegebene `\ID` ist:

`TRUE`, wenn:

- die Bahnaufzeichnung den Roboter mit `PathRecMoveBwd` rückwärts bewegt hat,
- der Roboter sich nicht von der Bahn entfernt hat, die mit `PathRecMoveBwd` abgearbeitet wird.

`FALSE`, wenn:

- die obigen Voraussetzungen nicht erfüllt sind.

Der Rückgabewert von `PathRecValidFwd`**mit** angegebener `\ID` ist:

`TRUE`, wenn:

- die Bahnaufzeichnung den Roboter mit `PathRecMoveBwd` rückwärts bewegt hat,
- der Roboter sich nicht von der Bahn entfernt hat, die mit `PathRecMoveBwd` abgearbeitet wird.
- die angegebene `\ID` während der Rückwärtsbewegung passiert wurde.

`FALSE`, wenn:

- die obigen Voraussetzungen nicht erfüllt sind.

Argumente

```
PathRecValidFwd ([\ID])
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.137 PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde

Path Recovery
Fortsetzung

`[\ID]`

Identifier

Datentyp: `pathrecid`

Variable, die den Namen der Aufzeichnungsstartposition angibt. Der Datentyp `pathrecid` weist keinen Wert auf (Non-Value) und wird nur als Bezeichner für die Benennung der Aufzeichnungsposition verwendet.

Programmabarbeitung

Nachdem die Bahnaufzeichnung mit `PathRecMoveBwd` zur Rückwärtsbewegung angewiesen wurde, kann geprüft werden, ob eine gültige aufgezeichnete Bahn zur Vorwärtsbewegung des Roboters vorhanden ist. Wenn der Bezeichner `\ID` weggelassen wird, gibt `PathRevValidFwd` zurück, ob eine Vorwärtsbewegung bis zu der Position möglich ist, an der die Rückwärtsbewegung initiiert wurde.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PathRecValidFwd`.

Beispiel 1:

```
VAR pathrecid id1;
VAR pathrecid id2;
VAR pathrecid id3;

PathRecStart id1;
MoveL p1, vmax, z50, tool1;
PathRecStart id2;
MoveL p2, vmax, z50, tool1;
PathRecStart id3;
!See figures 1 and 8 in tbe following table.
MoveL p3, vmax, z50, tool1;
ERROR
  StorePath;
  IF PathRecValidBwd(\ID:=id3) THEN
    !See figure 2 in the following table.
    PathRecMoveBwd \ID:=id3;
    ! Do some other operation
  ENDIF
  IF PathRecValidBwd(\ID:=id2) THEN
    !See figure 3 in the following table.
    PathRecMoveBwd \ID:=id2;
    ! Do some other operation
  ENDIF
  !See figure 4 in the following table.
  PathRecMoveBwd;
  ! Do final service action
  IF PathRecValidFwd(\ID:=id2) THEN
    !See figure 5 in the following table.
    PathRecMoveFwd \ID:=id2;
    ! Do some other operation
  ENDIF
```

Fortsetzung auf nächster Seite

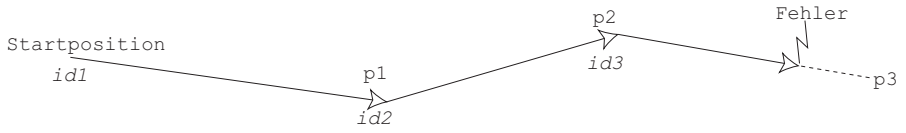
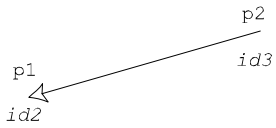
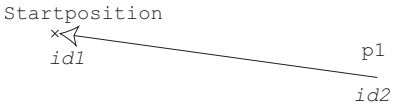
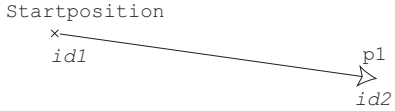
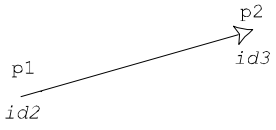
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.137 PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde

Path Recovery

Fortsetzung

```
IF PathRecValidFwd(\ID:=id3) THEN
  !See figure 6 in the following table.
  PathRecMoveFwd \ID:=id3;
  ! Do some other operation
ENDIF
!See figure 7 in the following table.
PathRecMoveFwd;
RestoPath;
StartMove;
RETRY;
```

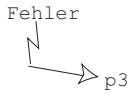
1	 xx0500002121
2	 xx0500002124
3	 xx0500002126
4	 xx0500002127
5	 xx0500002128
6	 xx0500002129
7	 xx0500002130

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.137 PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde

Path Recovery
Fortsetzung

8	 xx0500002131

Obiges Beispiel startet die Bahnaufzeichnung und fügt Bezeichner an drei Stellen der abgearbeiteten Bahn ein. Die obige Abbildung bezieht sich auf den Beispielcode und illustriert, wie der Roboter sich im Fall eines Fehlers bewegt, während die Abarbeitung zu Punkt `p3` fortschreitet. `PathRecValidBwd` bzw. `PathRecValidFwd` werden verwendet, da nicht im Voraus bestimmt werden kann, wo im Programm ein möglicher Fehler auftritt.

Syntax

```
PathRecValidFwd '('
  ['\' ID ':=' < variable (VAR) of pathrecid >] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Bezeichner der Bahnaufzeichnung	*pathrecid - Bezeichner der Bahnaufzeichnung auf Seite 1753*
Starten/Stoppen der Bahnaufzeichnung	*PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *PathRecStop - Stoppen der Bahnaufzeichnung auf Seite 531*
Prüfen, ob eine gültige Rückwärtsbahn existiert	*PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde auf Seite 1431*
Abspielen der Bahnaufzeichnung rückwärts	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*
Abspielen der Bahnaufzeichnung vorwärts	*PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen auf Seite 525*
Bewegung im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.138 PFRestart - Prüfen der unterbrochenen Bahn nach Stromausfall

Verwendung

`PFRestart` *(Power Failure Restart)* prüft, ob die Bahn beim Stromausfall unterbrochen wurde Wenn ja, sollten eventuell einige spezifische Aktionen ausgeführt werden. Die Funktion prüft die Bahn auf der aktuellen Ebene, Basisebene oder Interrupt-Ebene.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PFRestart`.

Beispiel 1:

```
IF PFRestart() = TRUE THEN
```

Es wird geprüft, ob in der aktuellen Ebene eine unterbrochene Bahn existiert. Wenn ja, gibt die Funktion `TRUE` zurück.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn eine unterbrochene Bahn in der angegebenen Ebene existiert, anderenfalls `FALSE`.

Argumente

```
PFRestart([\Base] | [\Irpt])
```

`[ \Base ]`

Base Level

Datentyp: `switch`

Gibt `TRUE` zurück, wenn in der Basisebene eine unterbrochene Bahn existiert.

`[ \Irpt ]`

Interrupt Level

Datentyp: `switch`

Gibt `TRUE` zurück, wenn auf der Ebene `StorePath` eine unterbrochene Bahn existiert.

Wenn kein Argument angegeben ist, gibt die Funktion `TRUE` zurück, falls auf der aktuellen Ebene eine unterbrochene Bahn existiert.

Syntax

```
PFRestart '('
  ['\' Base] | ['\' Irpt] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.139 PoseInv - Invertieren einer Koordinatenverschiebung

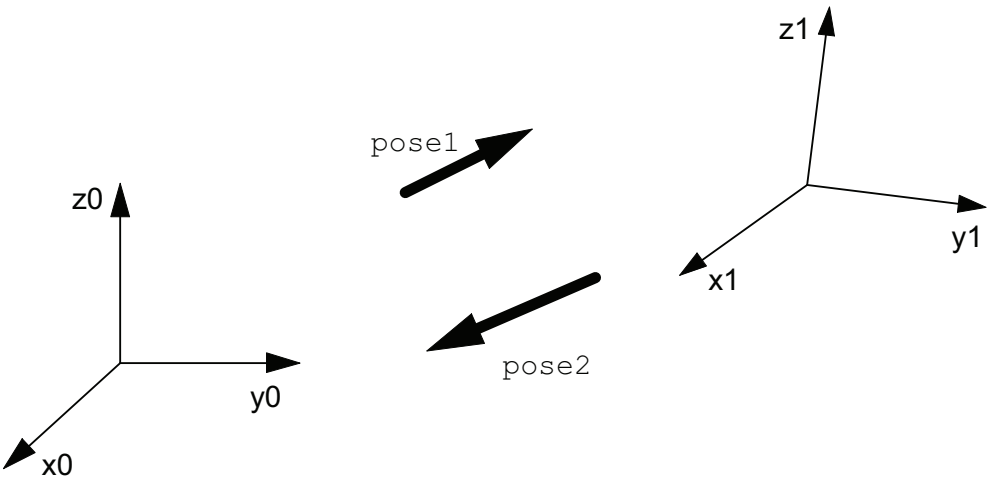
Verwendung

`PoseInv` (*Pose Invert*) berechnet die Umkehrung einer `pose`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PoseInv`.

Beispiel 1:



xx0500002443

`pose1` stellt das Koordinatensystem 1 in Bezug zu Koordinatensystem 0 dar. Die Umwandlung, die Koordinatensystem 0 in Bezug zu Koordinatensystem 1 ergibt, erfolgt mit der Umkehrung in `pose2`.

```
VAR pose pose1;
VAR pose pose2;
...
pose2 := PoseInv(pose1);
```

Rückgabewert

Datentyp: `pose`

Wert der invertierten Koordinatenverschiebung.

Argumente

```
PoseInv (Pose)
```

`Pose`

Datentyp: `pose`

Die zu invertierende `pose` (Koordinatenverschiebung).

Syntax

```
PoseInv'('
  [Pose ':='] <expression (IN) of pose>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.139 PoseInv - Invertieren einer Koordinatenverschiebung

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.140 PoseMult - Multiplizieren von Koordinatenverschiebungen

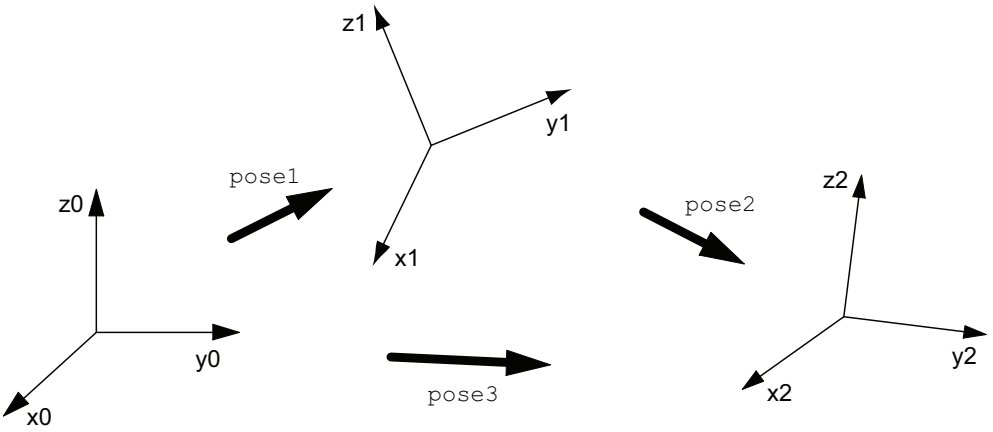
Verwendung

`PoseMult` (*Pose Multiply*) wird verwendet, um das Produkt zweier Koordinatenverschiebungen zu berechnen. Eine typische Verwendung ist die Berechnung einer neuen Position als Ergebnis einer Verschiebung, die auf eine ursprüngliche Position einwirkt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PoseMult`.

Beispiel 1:



xx0500002444

`pose1` repräsentiert das Koordinatensystem 1, das sich auf das Koordinatensystem 0 bezieht. `pose2` repräsentiert das Koordinatensystem 2, das sich auf das Koordinatensystem 1 bezieht. Die Transformation, die `pose3` ergibt, das Koordinatensystem 2 bezogen auf Koordinatensystem 0, wird durch das Produkt der beiden folgenden Transformationen erreicht:

```
VAR pose pose1;
VAR pose pose2;
VAR pose pose3;
...
pose3 := PoseMult(pose1, pose2);
```

Rückgabewert

Datentyp: `pose`

Der Wert des Produkts der beiden Koordinatenverschiebungen.

Argumente

```
PoseMult (Pose1 Pose2)
```

`Pose1`

Datentyp: `pose`

Die erste Koordinatenverschiebung.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.140 PoseMult - Multiplizieren von Koordinatenverschiebungen

RobotWare - OS

Fortsetzung

`Pose2`

Datentyp: `pose`

Die zweite Koordinatenverschiebung.

Syntax

```
PoseMult '('
  [Pose1 ':='] <expression (IN) of pose> ','
  [Pose2 ':='] <expression (IN) of pose> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pose`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.141 PoseVect - Anwenden einer Transformation auf einen Vektor

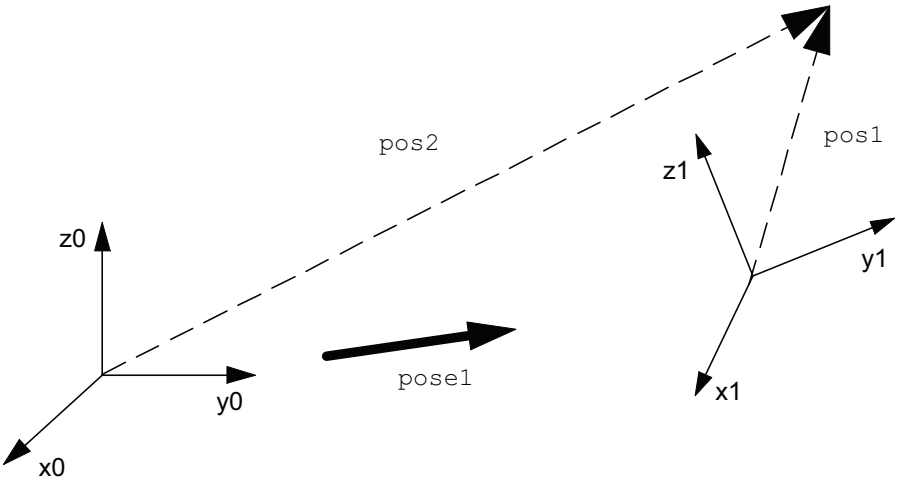
Verwendung

`PoseVect` (*Pose Vector*) wird verwendet, um das Produkt einer Koordinatenverschiebung und eines Vektors zu berechnen. Eine typische Verwendung ist die Berechnung eines Vektors, der das Ergebnis einer Verschiebung eines Originalvektors darstellt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PoseVect`.

Beispiel 1:



xx0500002445

`pose1` repräsentiert das Koordinatensystem 1, das sich auf das Koordinatensystem 0 bezieht.

`pos1` ist ein Vektor mit Bezug zu Koordinatensystem 1. Der entsprechende Vektor mit Bezug zu Koordinatensystem 0 wird durch das Produkt berechnet

```
VAR pose pose1;
VAR pos pos1;
VAR pos pos2;
...
...
pos2:= PoseVect(pose1, pos1);
```

Rückgabewert

Datentyp: `pos`

Der Wert des Produkts der Koordinatenverschiebung und der Originalposition.

Argumente

```
PoseVect (Pose Pos)
```

`Pose`

Datentyp: `pose`

Die zuzuweisende Transformation.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.141 PoseVect - Anwenden einer Transformation auf einen Vektor

RobotWare - OS

Fortsetzung

`Pos`

Datentyp: `pos`

Die zu transformierende Position.

Syntax

```
PoseVect '('
  [Pose ':='] <expression (IN) of pose> ','
  [Pos ':='] <expression (IN) of pos> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `pos`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.142 Pow - Berechnen eines Potenzwerts

Verwendung

Pow (*Power*) wird verwendet, um den Exponentialwert zu einer beliebigen Basis zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion Pow.

Beispiel 1:

```
VAR num x;
VAR num y
VAR num reg1;
...
reg1:= Pow(x, y);
```

reg1 wird der Wert x^y zugewiesen.

Rückgabewert

Datentyp: num

Der Wert der Base mit dem Exponenten, d. h. Basis^{Exponent}.

Argumente

```
Pow (Base Exponent)
```

Base

Datentyp: num

Der Wert des Basisarguments.

Exponent

Datentyp: num

Der Wert des Exponentenarguments.

Einschränkungen

Die Abarbeitung der Funktion x^y ergibt einen Fehler, wenn:

- x < 0 und y keine Ganzzahl ist;
- x = 0 und y ≤ 0.

Syntax

```
Pow '('
  [Base ':='] <expression (IN) of num> ','
  [Exponent ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.143 PowDnum - Berechnen eines Potenzwerts

Verwendung

PowDnum (*Power Dnum)* wird verwendet, um den Exponentialwert zu einer beliebigen Basis zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion PowDnum.

Beispiel 1:

```
VAR dnum x;
VAR num y
VAR dnum value;
...
value:= PowDnum(x, y);
```

value wird der Wert x^y zugewiesen.

Rückgabewert

Datentyp: dnum

Der Wert der Base mit dem Exponenten, d. h. Basis^{Exponent}.

Argumente

```
PowDnum (Base Exponent)
```

Base

Datentyp: dnum

Der Wert des Basisarguments.

Exponent

Datentyp: num

Der Wert des Exponentenarguments.

Einschränkungen

Die Abarbeitung der Funktion x^y ergibt einen Fehler, wenn:

- x < 0 und y keine Ganzzahl ist;
- x = 0 und y ≤ 0.

Syntax

```
PowDnum '('
  [Base ':='] <expression (IN) of dnum> ','
  [Exponent ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs dnum.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.144 PPMovedInManMode - Überprüfen, ob der Programmzeiger im Einrichtbetrieb verschoben wurde

Verwendung

`PPMovedInManMode` gibt TRUE zurück, wenn der Benutzer den Programmzeiger verschoben hat, während sich die Steuerung im Einrichtbetrieb befand, d. h. die Bedienertaste war in der Stellung Einrichtbetrieb bei reduzierter Geschwindigkeit oder Einrichtbetrieb bei voller Geschwindigkeit. Der Status des verschobenen Programmzeigers wird mit der Instruktion `ResetPPMoved` oder durch Einstellen der Taste von Automatikbetrieb auf Einrichtbetrieb zurückgesetzt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `PPMovedInManMode`.

Beispiel 1:

```
IF PPMovedInManMode() THEN
  WarnUserOfPPMovement;
  DoJob;
ELSE
  DoJob;
ENDIF
```

Rückgabewert

Datentyp: `bool`

TRUE, wenn der Programmzeiger vom Benutzer im Einrichtbetrieb verschoben wurde

Programmabarbeitung

Überprüfen, ob der Programmzeiger für die aktuelle Programmtask im Einrichtbetrieb verschoben wurde.

Syntax

```
PPMovedInManMode '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Überprüfen, ob Programmzeiger verschoben wurde	*IsStopStateEvent – Prüfen, ob Programmzeiger bewegt wurde auf Seite 1383*
Zurücksetzen des Status für den im Einrichtbetrieb verschobenen Programmzeiger	*ResetPPMoved - Zurücksetzen des Status für den im Einrichtbetrieb verschobenen Programmzeiger auf Seite 617*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.145 Present - Prüft, ob ein optionaler Parameter verwendet wird

Verwendung

`Present` wird verwendet, um zu prüfen, ob ein optionales Argument beim Aufruf einer Routine verwendet wurde.

Ein optionaler Parameter kann nicht verwendet werden, wenn er beim Routinenaufruf nicht angegeben war. Mit dieser Funktion können Sie prüfen, ob ein Parameter angegeben war, um das Auftreten von Fehlern zu vermeiden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Present`.

Siehe auch *Weitere Beispiele auf Seite 1448*.

Beispiel 1:

```
PROC feeder (\switch on | switch off)
  IF Present (on) Set do1;
  IF Present (off) Reset do1;
ENDPROC
```

Der Ausgang `do1`, der eine Zuführung steuert, wird gesetzt oder zurückgesetzt, abhängig davon, welches Argument beim Aufruf der Routine verwendet wurde.

Rückgabewert

Datentyp: `bool`

TRUE = Der Parameterwert oder ein Schalter wurde beim Aufruf der Routine definiert.

FALSE = Der Parameterwert oder ein Schalter wurde nicht definiert.

Argumente

```
Present (OptPar)
```

`OptPar`

Optional Parameter

Datentyp: `anytype`

Der Name des zu prüfenden optionalen Parameters.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Present`.

Beispiel 1:

```
PROC glue (\switch on, num glueflow, robtarget topoint, speeddata
     speed, zonedata zone, PERS tooldata tool, \PERS wobjdata wobj)
  IF Present (on) PulseDO glue_on;
    SetAO gluesignal, glueflow;
  IF Present (wobj) THEN
    MoveL topoint, speed, zone, tool \WObj:=wobj;
  ELSE
    MoveL topoint, speed, zone, tool;
  ENDIF
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
ENDPROC
```

Eine Kleberoutine wird erstellt. Wenn das Argument `\on` beim Aufruf der Routine angegeben ist, wird ein Impuls auf dem Signal `glue_on` generiert. Der Roboter setzt dann den analogen Ausgang `gluesignal`, der die Klebepistole steuert und bewegt sich an die Endposition. Da „wobj“ ein optionaler Parameter ist, werden abhängig von der Verwendung dieses Arguments unterschiedliche `MoveL`-Instruktionen verwendet.

Syntax

```
Present '('
  [OptPar ':='] <reference (REF) of anytype> ')'
```

Ein REF-Parameter verlangt in diesem Fall den Namen des optionalen Parameters.

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Routineparameter	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.146 ProgMemFree - Abrufen der Größe des freien Programmspeichers

Verwendung

`ProgMemFree` (*Program Memory Free*) wird zum Abrufen der Größe des freien Programmspeichers verwendet.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `ProgMemFree`.

Beispiel 1:

```
FUNC num module_size(string file_path)
  VAR num pgmfree_before;
  VAR num pgmfree_after;

  pgmfree_before:=ProgMemFree();
  Load \Dynamic, file_path;
  pgmfree_after:=ProgMemFree();
  Unload file_path;
  RETURN (pgmfree_before-pgmfree_after);
ENDFUNC
```

`ProgMemFree` wird in einer Funktion benutzt, die den Wert für die Größe des Speichers zurückgibt, den ein Modul im Programmspeicher zuteilt.

Rückgabewert

Datentyp: `num`

Die Größe des freien Programmspeichers in Byte.

Syntax

```
ProgMemFree '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Laden eines Programmmoduls	*Load - Laden eines Programmmoduls während der Abarbeitung auf Seite 346*
Entfernen eines Programmmoduls	*UnLoad - Entladen eines Programmmoduls während der Abarbeitung auf Seite 1030*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.147 PrxGetMaxRecordpos - Ruft die maximale Sensorposition ab

Verwendung

PrxGetMaxRecordpos wird dazu verwendet, die maximale Position in mm der aktiven Aufzeichnung zurückzugeben.

Die maximale Sensorposition kann zur Skalierung oder Begrenzung des Arguments `max_sync` in der Instruktion `SyncToSensor` verwendet werden.

Grundlegendes Beispiel

```
maxpos:=PrxGetMaxRecordpos Ssync1;
```

Ruft die maximale Sensorposition des aktiven Profils der mechanischen Einheit `Ssync1` ab.

Rückgabewert

Datentyp: `num`

Die maximale Position (in mm) des aufgezeichneten Profils der Sensorbewegung.

Argumente

```
PrxGetMaxRecordpos MechUnit
```

`MechUnit`

Datentyp: `mechunit`

Das Objekt der sich bewegenden mechanischen Einheit, auf das die Roboterposition synchronisiert wird.

Programmabarbeitung

Die Aufzeichnung muss abgeschlossen und die Aufzeichnung aktiv sein.

Syntax

```
PrxGetMaxRecordpos '('
  [ MechUnit ':=' ] < expression (IN) of mechunit> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Machine Synchronization	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.148 Rand – Erzeugen Sie eine Zufallszahl

Verwendung

Rand wird verwendet, um eine zufällige Ganzzahl zwischen 0 und RAND_MAX zu erzeugen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion Rand.

Beispiel 1:

```
VAR num myrandomnumber;
..
myrandomnumber:=Rand();
```

Erzeugen Sie Zufallszahl und weisen Sie diese der Variablen myrandomnumber zu.

Rückgabewert

Datentyp: num

Eine Zufallszahl zwischen 0 und 65535 (0 und 32767 für eine virtuelle Steuerung).

Argumente

```
Rand ([\Seed] [\SeedEachNTime])
```

\Seed

Datentyp: switch

Die aktuelle Zeit wird für den Seed des Zufallszahlgenerators verwendet, wenn dieser Switch verwendet wird. Wenn der Switch nicht verwendet wird, wird mit dem Seed zuerst die Funktion Rand aufgerufen. Die Sequenz der erzeugten Zahlen wird immer identisch sein.

\SeedEachNTime

Datentyp: num

Legt einen internen Zähler fest, der jedes Mal verringert wird, wenn die Funktion Rand mit dem optionalen Argument SeedEachNTime abgearbeitet wird. Wenn der interne Zähler 0 ist, erfolgt eine neue Sequenz (mit aktueller Zeit), und der interne Zähler wird auf den Wert gesetzt, der im optionalen Argument SeedEachNTime verwendet wird.

Programmabarbeitung

Rand dient zum Erzeugen einer zufälligen ganzen Zahl zwischen 0 und RAND_MAX.

In der Realität sind pseudozufällige Zahlen überhaupt nicht zufällig. Die Zahlen werden mit einem festen deterministischen Algorithmus berechnet. Die *Sequenz* (Argument Seed) ist der Startpunkt für eine Sequenz von Zufallszahlen. Beim Starten von derselben Sequenz erhalten Sie dieselbe Sequenz.

Die Funktion Rand verwendet die aktuelle Zeit, wenn sie zum ersten Mal ausgeführt wird. Um sie zufälliger zu gestalten, ist es möglich, festzulegen, dass sie bei jeder

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Ausführung eine neue Zeit verwenden soll (mit dem Schalter `Seed`), oder nach einer Mehrfachausführung (optionales Argument `SeedEachNTime`).

Wenn Sie niemals einen neuen Startpunkt (Zeit) für die Zufallszahlen haben und viele Zahlen mit `Rand` generieren, sehen Sie, dass dieselbe Zahlenfolge wiederholt wird.

Weitere Beispiele

Weitere Beispiele für die Funktion `Rand` sind unten aufgeführt.

Beispiel 1:

```
VAR num random_numbers{6};
FOR i FROM 1 TO 6 DO
  random_numbers{i} := Rand()/RAND_MAX;
ENDFOR
```

Im obigen Beispiel erzeugt die Funktion `Rand` 6 Zufallszahlen zwischen 0 und 1 und speichert sie im Datenfeld `random_numbers`.

Beispiel 2:

```
VAR num rand_no;
Open "HOME:" \File:= "LOGFILE.txt", logfile \Write;
FOR i FROM 1 TO 5000 DO
  rand_no := Rand(\SeedEachNTime:=30);
  Write logfile, ""\Num:=rand_no;
ENDFOR
```

Im obigen Beispiel `Rand` erzeugt die Funktion 5000 Zufallszahlen und schreibt sie in eine Datei. Das optionale Argument `SeedEachNTime` wird mit dem Wert 30 verwendet. Dann wird eine neue Auswahl durchgeführt, um nach jeden 30-mal Ausführen der eine Funktion `Rand` neue Zufallszahlreihe zu erstellen.

Syntax

```
Rand '('
  ['\' Seed]
  ['\' SeedEachNTime':=' <expression (IN) of num> ]')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.149 RawBytesLen - Abrufen der Länge von rawbytes-Daten

Verwendung

RawBytesLen wird verwendet, um die aktuelle Länge der gültigen Bytes in einer rawbytes-Variablen abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion RawBytesLen.

Beispiel 1:

```
VAR rawbytes from_raw_data;
VAR rawbytes to_raw_data;
VAR num integer := 8
VAR num float := 13.4;
ClearRawBytes from_raw_data;
PackRawBytes integer, from_raw_data, 1 \IntX := INT;
PackRawBytes float, from_raw_data, (RawBytesLen(from_raw_data)+1)
     \Float4;
CopyRawBytes from_raw_data, 1, to_raw_data, 3;
```

In diesem Beispiel wird die Variable from_raw_data des Typs „rawbites“ zunächst gelöscht, d. h., alle Byte werden auf 0 gesetzt (wie der Standardwert bei Deklaration). Dann wird in die ersten beiden Byte der Integer-Wert übernommen und mithilfe der Funktion RawBytesLen wird der Wert float in die nächsten 4 Byte übernommen (beginnend bei Index 3).

Nachdem from_raw_data mit Daten gefüllt wurde, wird der Inhalt (6 Byte) beginnend mit Position 3 in to_raw_data kopiert.

Rückgabewert

Datentyp: num

Die aktuelle Länge gültiger Byte in einer Variablen des Typs rawbytes; Bereich 0 ... 1024.

Im Allgemeinen wird die aktuelle Länge gültiger Bytes in einer rawbytes-Variablen durch das System als letztes geschriebenes Byte in der rawbytes-Struktur aktualisiert.

Weitere Informationen finden Sie im Datentyp rawbytes, Instruktion ClearRawBytes, CopyRawBytes, PackDNHeader, PackRawBytes und ReadRawBytes.

Argumente

```
RawBytesLen (RawData)
```

RawData

Datentyp: rawbytes

RawData ist der Datencontainer, dessen aktuelle Länge an gültigen Bytes zurückgegeben werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Während der Programmabarbeitung wird die aktuelle Länge an gültigen Bytes zurückgegeben.

Syntax

```
RawBytesLen '('
[RawData ':='] < variable (VAR) of rawbytes> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
`rawbytes`-Daten	*rawbytes – Unverarbeitete Daten auf Seite 1764*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.150 ReadBin - Liest ein Bytes aus einer Datei oder einem E/A-Gerät

Verwendung

`ReadBin` (*Read Binary*) wird verwendet, um ein Byte (8 Bits) aus einer Datei oder einem E/A-Gerät zu lesen.

Diese Funktion eignet sich sowohl für binäre als auch zeichenorientierte Dateien oder E/A-Geräte.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ReadBin`.

Siehe auch *Weitere Beispiele auf Seite 1457*.

Beispiel 1:

```
VAR num character;
VAR iodev file1;
...
Open "HOME:", \File:= "FILE1.DOC", file1\Bin;
character := ReadBin(file1);
```

Ein Byte wird aus der binären Datei `file1` gelesen.

Rückgabewert

Datentyp: `num`

Ein Byte (8 Bits) wird aus der angegebenen Datei oder dem E/A-Gerät gelesen. Dieses wird in den entsprechenden positiven numerischen Wert konvertiert und als Datentyp `num` zurückgegeben. Wenn die Datei leer ist (Dateiende), wird `EOF_BIN` (die Zahl -1) zurückgegeben.

Argumente

```
ReadBin (IODevice [\Time])
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) der Datei oder des E/A-Geräts, die/der gelesen werden soll.

`[\Time]`

Datentyp: `num`

Die maximale Dauer für den Lesevorgang (Timeout) in Sekunden. Wenn dieses Argument nicht angegeben wird, beträgt die maximale Dauer 60 Sekunden. Verwenden Sie für eine unendliche Wartezeit die vordefinierte Konstante `WAIT_MAX`.

Wenn diese Zeit abläuft, bevor der Lesevorgang abgeschlossen ist, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_DEV_MAXTIME`. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Die Timeout-Funktion ist auch während des Programmstopps in Gebrauch und wird vom RAPID-Programm beim Programmstart berücksichtigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Die Programmabarbeitung wartet, bis ein Byte (8 Bits) aus der angegebenen Datei oder dem E/A-Gerät gelesen werden kann.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Vordefinierte Daten

Mithilfe der Konstanten `EOF_BIN` kann der Lesevorgang am Ende der Datei gestoppt werden.

```
CONST num EOF_BIN := -1;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Lesen.
`ERR_DEV_MAXTIME`	Timeout, bevor der Lesevorgang beendet ist.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ReadBin`.

Beispiel 1:

```
VAR num bindata;
VAR iodev file;

Open "HOME:/myfile.bin", file \Read \Bin;
bindata := ReadBin(file);
WHILE bindata <> EOF_BIN DO
  TPWrite ByteToStr(bindata\Char);
  bindata := ReadBin(file);
ENDWHILE
```

Liest den Inhalt der binären Datei "myfile.bin" von Beginn bis Ende und zeigt die empfangenen binären Daten am FlexPendant in Zeichen konvertiert an (ein Zeichen pro Zeile).

Einschränkungen

Diese Funktion kann nur für Dateien oder E/A-Geräte verwendet werden, die mit Lesezugriff geöffnet wurden (`\Read` für zeichenorientierte Dateien, `\Bin` oder `\Append \Bin` für Binärdateien).

Syntax

```
ReadBin '('
  [IODevice ':='] <variable (VAR) of iodev>
  ['\' Time ':=' <expression (IN) of num>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.150 ReadBin - Liest ein Bytes aus einer Datei oder einem E/A-Gerät

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Dateien, E/A-Geräte öffnen etc.	*Technisches Referenzhandbuch - RAPID Overview*
Konvertieren von byte-Daten in string-Daten	*ByteToStr - Konvertieren von byte-Daten in string-Daten auf Seite 1218*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.151 ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis

Verwendung

`ReadDir` wird verwendet, um den Namen der nächsten Datei oder des nächsten Unterverzeichnisses unter einem Verzeichnis abzurufen, das mit der Instruktion `OpenDir` geöffnet wurde.

Solange die Funktion den Wert `TRUE` zurückgibt, können weitere Dateien bzw. Unterverzeichnisse abrufbar sein.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ReadDir`.

Siehe auch *Weitere Beispiele auf Seite 1460*.

Beispiel 1:

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
  CloseDir directory;
ENDPROC
```

Dieses Beispiel gibt die Namen aller Dateien bzw. Unterverzeichnisse unter dem angegebenen Verzeichnis aus.

Rückgabewert

Datentyp: `bool`

Die Funktion gibt `TRUE` zurück, wenn sie einen Namen abgerufen hat, anderenfalls gibt sie `FALSE` zurück.

Argumente

```
ReadDir (Dev FileName)
```

`Dev`

Datentyp: `dir`

Eine Variable mit Verweis auf das Verzeichnis, abgerufen durch die Instruktion `OpenDir`.

`FileName`

Datentyp: `string`

Name der abgerufenen Datei bzw. des abgerufenen Unterverzeichnisses.

Programmabarbeitung

Diese Funktion gibt einen booleschen Wert zurück, der angibt, ob der Abruf eines Namens erfolgreich war.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.151 ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_FILEACC	Das Verzeichnis wird nicht geöffnet (siehe OpenDir). Der gelesene Dateiname enthält mehr Bytes als die maximale Länge eines RAPID-Strings. Die im Argument FileName verwendete Stringvariable wird nicht aktualisiert.

Weitere Beispiele

Weitere Beispiele für die Funktion ReadDir sind unten aufgeführt

Beispiel 1:

Dieses Beispiel implementiert einen generischen Durchlauf einer Verzeichnisstrukturfunktion.

```
PROC searchdir(string dirname, string actionproc)
  VAR dir directory;
  VAR string filename;
  IF IsFile(dirname \Directory) THEN
    OpenDir directory, dirname;
    WHILE ReadDir(directory, filename) DO
        ! .. and . is the parent and resp. this directory
        IF filename <> ".." AND filename <> "." THEN
          searchdir dirname+"/"+filename, actionproc;
        ENDIF
    ENDWHILE
    CloseDir directory;
  ELSE
    %actionproc% dirname;
  ENDIF
ERROR
  RAISE;
ENDPROC

PROC listfile(string filename)
  TPWrite filename;
ENDPROC

PROC main()
  ! Execute the listfile routine for all files found under the
  ! tree in HOME:
  searchdir "HOME:","listfile";
ENDPROC
```

Dieses Programm durchläuft die Verzeichnisstruktur unter HOME: und ruft für jede gefundene Datei die Prozedur listfile auf. Die Angabe searchdir ist der generische Teil, der weder den Beginn der Suche noch die aufzurufende Routine für jede Datei kennt. Sie verwendet IsFile, um zu prüfen, ob ein Unterverzeichnis oder eine Datei gefunden wurde, und verwendet den Mechanismus der späten

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Bindung, um für alle gefundenen Dateien die Prozedur aufzurufen, die in `actionproc` angegeben ist. Die Routine `actionproc` sollte eine Prozedur mit einem Parameter des Typs `string` sein.

Syntax

```
ReadDir '('
  [ Dev ':=' ] < variable (VAR) of dir> ','
  [ FileName ':=' ] < var or pers (INOUT) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Verzeichnis	*dir – Dateiverzeichnisstruktur auf Seite 1689*
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Öffnen eines Verzeichnisses	*OpenDir - Öffnen eines Verzeichnisses auf Seite 497*
Schließen eines Verzeichnisses	*CloseDir - Schließen eines Verzeichnisses auf Seite 153*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.152 ReadMotor - Liest die aktuellen Motorwinkel.

Verwendung

`ReadMotor` wird zum Lesen der aktuellen Winkel der verschiedenen Motoren des Roboters und der externen Achsen verwendet. Primär wird diese Funktion in der Kalibrierungsprozedur des Roboters eingesetzt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ReadMotor`.

Siehe auch *Weitere Beispiele auf Seite 1463*.

Beispiel 1:

```
VAR num motor_angle2;
motor_angle2 := ReadMotor(2);
```

Der aktuelle Motorwinkel der zweiten Roboterachse wird in `motor_angle2` gespeichert.

Rückgabewert

Datentyp: `num`

Der aktuelle Motorwinkel in Radiant der angegebenen Roboterachse oder der externen Achsen.

Argumente

```
ReadMotor [\MecUnit ] Axis
```

`MecUnit`

Mechanical Unit

Datentyp: `mecunit`

Der Name der mechanischen Einheit, für die eine Achse gelesen werden soll. Wenn dieses Argument weggelassen wird, wird die Achse für den angeschlossenen Roboter gelesen.

`Axis`

Datentyp: `num`

Die Nummer der zu lesenden Achse (1 bis 6).

Programmabarbeitung

Der zurückgegebene Motorwinkel entspricht der aktuellen Position in Radiant für den Motor ohne Kalibrierungsversatz. Der Wert bezieht sich nicht auf eine feste Roboterposition, sondern nur auf die Resolver-interne Nullposition, d. h. normalerweise auf die Resolver-Nullposition, die der Kalibrierposition am nächsten ist (die Differenz zwischen der Resolver-Nullposition und der Kalibrierposition ist der Wert des Kalibrierungsversatzes). Der Wert stellt die volle Bewegung jeder Achse dar, auch wenn es sich um mehrere Umdrehungen handelt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_AXIS_PAR	Parameter "Achse" in der Funktion ist falsch.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion ReadMotor.

Beispiel 1:

```
VAR num motor_angle;
motor_angle := ReadMotor(\MecUnit:=STN1, 1);
```

Der aktuelle Motorwinkel der ersten Achse von STN1 wird in motor_angle gespeichert.

Einschränkungen

Es ist nur möglich, die aktuellen Motorwinkel der mechanischen Einheiten zu lesen, die vom aktuellen Programmtask gesteuert werden. Für einen Nicht-Bewegungstask ist es möglich, die Motorwinkel der mechanischen Einheiten zu lesen, die vom verknüpften Bewegungstask gesteuert werden.

Syntax

```
ReadMotor '('
  ['\' MecUnit ':=' < variable (VAR) of mecunit> ',']
  [Axis ':=' ] < expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Lesen des aktuellen Achsenwinkels	*CJointT - Lesen der aktuellen Achsenwinkel auf Seite 1248*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.153 ReadNum - Liest eine Zahl aus einer Datei oder einem E/A-Gerät

Verwendung

`ReadNum` (*Read Numeric*) wird verwendet, um eine Zahl aus einer zeichenorientierten Datei oder einem zeichenorientierten E/A-Gerät zu lesen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `ReadNum`.

Siehe auch *Weitere Beispiele auf Seite 1465*.

Beispiel 1:

```
VAR iodev infile;
...
Open "HOME:/file.doc", infile\Read;
reg1 := ReadNum(infile);
```

`reg1` wird eine gelesene Zahl aus der Datei `file.doc` zugewiesen.

Rückgabewert

Datentyp: `num`

Der numerische Wert, der aus der angegebenen Datei oder dem E/A-Gerät gelesen wurde. Wenn die Datei leer ist (Dateiende), wird eine Zahl größer als `EOF_NUM` `(9.998E36)` zurückgegeben.

Argumente

```
ReadNum (IODevice [\Delim] [\Time])
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) der Datei oder des E/A-Geräts, die/der gelesen werden soll.

`[\Delim]`

Delimiters

Datentyp: `string`

Eine Zeichenfolge mit Begrenzern für die Analyse einer Zeile in der Datei oder dem E/A-Gerät. Standardmäßig (ohne `\Delim`) wird die Datei Zeile für Zeile gelesen und ein Zeilenvorschubzeichen (\0A) wird als einziger Begrenzer bei der Analyse berücksichtigt. Wenn das Argument `\Delim` verwendet wird, wird jedes Zeichen im angegebenen string-Argument als signifikanter Teil der Zeile betrachtet.

Bei Verwendung des Arguments `\Delim` fügt die Steuerung stets das Wagenrücklauf- (`\0D`) und das Zeilenvorschubzeichen (`\0A`) den Begrenzern hinzu, die der Anwender angegeben hat.

Um nicht alphanumerische Zeichen anzugeben, verwenden Sie `\xx`, wobei `xx` die hexadezimale Darstellung des ASCII-Codes für das Zeichen ist (Beispiel: TAB wird durch `\09` angegeben).

`[\Time]`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die maximale Dauer für den Lesevorgang (Timeout) in Sekunden. Wenn dieses Argument nicht angegeben wird, beträgt die maximale Dauer 60 Sekunden. Verwenden Sie für eine unendliche Wartezeit die vordefinierte Konstante `WAIT_MAX`.

Wenn diese Zeit abläuft, bevor der Lesevorgang abgeschlossen ist, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_DEV_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Die Timeout-Funktion ist auch während des Programmstopps in Gebrauch und wird vom RAPID-Programm beim Programmstart benachrichtigt.

Programmabarbeitung

Die Funktion liest ab der aktuellen Dateiposition und verwirft alle führenden Begrenzer. Ein führender Begrenzer ohne das Argument `\Delim` ist ein Zeilenvorschubzeichen. Führende Begrenzer mit dem Argument `\Delim` sind beliebige Zeichen, die im Argument `\Delim` angegeben sind, sowie Wagenrücklauf- und Zeilenvorschubzeichen. Dann werden sämtliche Zeichen bis einschließlich des nächsten Begrenzers (der verworfen wird) gelesen, jedoch nicht mehr als 80 Zeichen. Wenn der signifikante Teil 80 Zeichen überschreitet, werden die übrigen Zeichen beim nächsten Lesevorgang übernommen.

Die gelesene Zeichenfolge wird dann in einen numerischen Wert konvertiert, z. B. wird "234.4" in den numerischen Wert 234,4 konvertiert.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Vordefinierte Daten

Mithilfe der Konstanten `EOF_NUM` kann der Lesevorgang am Ende der Datei gestoppt werden.

```
CONST num EOF_NUM := 9.998E36;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Zugriffsfehler beim Lesen.
`ERR_RCVDATA`	Es wird versucht, nicht-numerische Daten zu lesen.
`ERR_DEV_MAXTIME`	Timeout, bevor der Lesevorgang beendet ist.

Weitere Beispiele

Folgende Beispiele veranschaulichen die Funktion `ReadNum`.

Beispiel 1:

```
reg1 := ReadNum(infile\Delim:="\09");
IF reg1 > EOF_NUM THEN
  TPWrite "The file is empty";
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.153 ReadNum - Liest eine Zahl aus einer Datei oder einem E/A-Gerät

RobotWare - OS

Fortsetzung

```
...
```

Liest eine Zahl in einer Zeile, in der Zahlen durch TABULATOR- (`"\09"`) oder LEERZEICHEN (`" "`) getrennt sind. Vor dem Lesen der Zahl aus der Datei wird geprüft, ob die Datei leer ist.

> **Hinweis**
>
> Verwenden Sie < oder > („größer als“ oder „kleiner als“), um zu überprüfen, ob die Datei leer ist. Verwenden Sie nicht „=“ (ist gleich).

Einschränkungen

Diese Funktion kann nur für zeichenorientierte Dateien verwendet werden, die zum Lesen geöffnet wurden.

Syntax

```
ReadNum '('
  [IODevice ':='] <variable (VAR) of iodev>
  ['\' Delim ':=' <expression (IN) of string>]
  ['\' Time ':=' <expression (IN) of num>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Dateien, E/A-Geräte öffnen etc.	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.154 ReadStr - Liest eine Zeichenfolge aus einer Datei oder einem E/A-Gerät

Verwendung

`ReadStr` (*Read String*) wird verwendet, um eine Zeichenfolge aus einer zeichenorientierten Datei oder einem zeichenorientierten E/A-Gerät zu lesen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `ReadStr`.

Siehe auch *Weitere Beispiele auf Seite 1469*.

Beispiel 1:

```
VAR string text;
VAR iodev infile;
...
Open "HOME:/file.doc", infile\Read;
text := ReadStr(infile);
```

`text` wird eine gelesene Zeichenfolge aus der Datei `file.doc` zugewiesen.

Rückgabewert

Datentyp: `string`

Die Zeichenfolge, die aus der angegebenen Datei oder dem seriellen Kanal gelesen wurde. Wenn die Datei leer ist (Dateiende), wird die Zeichenfolge `"EOF"` zurückgegeben.

Argumente

```
ReadStr (IODevice [\Delim] [\RemoveCR] [\DiscardHeaders] [\Time]
     [\Line])
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) der Datei oder des E/A-Geräts, die/der gelesen werden soll.

`[\Delim]`

Delimiters

Datentyp: `string`

Eine Zeichenfolge mit Begrenzern für die Analyse einer Zeile in der Datei oder dem E/A-Gerät. Standardmäßig wird die Datei Zeile für Zeile gelesen und ein Zeilenvorschubzeichen (\0A) wird als einziger Begrenzer bei der Analyse berücksichtigt. Wenn das Argument `\Delim` verwendet wird, wird jedes Zeichen im angegebenen Zeichenfolge-Argument sowie standardmäßig auch Zeilenvorschubzeichen als signifikanter Teil der Zeile betrachtet.

Um nicht alphanumerische Zeichen anzugeben, verwenden Sie `\xx` , wobei `xx` die hexadezimale Darstellung des ASCII-Codes für das Zeichen ist (Beispiel: TAB wird durch `\09` angegeben).

`[\RemoveCR]`

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.154 ReadStr - Liest eine Zeichenfolge aus einer Datei oder einem E/A-Gerät
RobotWare - OS
Fortsetzung

Ein Schalter zum Entfernen des abschließenden CR-Zeichens (Wagenrücklauf) beim Lesen von PC-Dateien. In PC-Dateien wird eine neue Zeile durch einen Wagenrücklauf und einen Zeilenvorschub (CRLF) angegeben. Beim Lesen einer Zeile in solchen Dateien wird das Wagenrücklaufzeichen standardmäßig in die zurückgegebene Zeichenfolge eingelesen. Mit diesem Argument wird das Wagenrücklaufzeichen zwar aus der Datei gelesen, aber nicht in die zurückgegebene Zeichenfolge aufgenommen.

`[\DiscardHeaders]`

Datentyp: `switch`

Dieses Argument gibt an, ob die führenden Begrenzer (angegeben in `\Delim` plus Standard-Zeilenvorschub) übersprungen werden, bevor die Daten an die Rückgabe-Zeichenfolge übertragen werden. Wenn das erste Zeichen an der aktuellen Dateiposition ein Begrenzer ist, wird es standardmäßig gelesen, aber nicht an die Rückgabe-Zeichenfolge übertragen; die Zeilenanalyse wird gestoppt und eine leere Zeichenfolge wird zurückgegeben. Wenn dieses Argument verwendet wird, werden alle Begrenzer in der Zeile aus der Datei gelesen, aber ignoriert, und es wird kein Wert zurückgegeben, bis die Rückgabezeichenfolge die Daten enthält, die mit dem ersten Zeichen in der Zeile beginnen, das kein Begrenzer ist.

`[\Time]`

Datentyp: `num`

Die maximale Dauer für den Lesevorgang (Timeout) in Sekunden. Wenn dieses Argument nicht angegeben wird, beträgt die maximale Dauer 60 Sekunden. Verwenden Sie für eine unendliche Wartezeit die vordefinierte Konstante `WAIT_MAX`.

Wenn diese Zeit abläuft, bevor der Lesevorgang abgeschlossen ist, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_DEV_MAXTIME` aufgerufen. Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Die Timeout-Funktion ist auch während des Programmstopps in Gebrauch und wird im RAPID-Programm beim Programmstart berücksichtigt.

`[\Line]`

Datentyp: `num`

Legt fest, welche Zeile in der Datei gelesen werden soll.

Wenn die Zeile nicht existiert, wird die Zeichenfolge `"EOF"` zurückgegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Wenn das Argument `\DiscardHeaders` verwendet wird, liest und verwirft die Funktion ab der aktuellen Dateiposition alle führenden Begrenzer (Zeilenvorschubzeichen sowie alle Zeichen, die im Argument `\Delim` angegeben sind). In allen Fällen werden dann sämtliche Zeichen bis zum nächsten Begrenzungszeichen gelesen, jedoch nicht mehr als 80 Zeichen. Wenn der signifikante Teil 80 Zeichen überschreitet, werden die übrigen Zeichen beim nächsten Lesevorgang übernommen. Der Begrenzer, der den Analysestopp verursacht hat, wird aus der Datei gelesen, aber nicht an die Rückgabe-Zeichenfolge übertragen. Wenn das letzte Zeichen in der Zeichenfolge ein Wagenrücklaufzeichen ist und das Argument `\RemoveCR` verwendet wird, wird dieses Zeichen nicht aus der Zeichenfolge entfernt.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Vordefinierte Daten

Mit der Konstante `EOF` kann geprüft werden, ob die Datei beim Leseversuch leer war. Mit ihr kann auch der Lesevorgang am Ende der Datei gestoppt werden.

```
CONST string EOF := "EOF";
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Lesen.
`ERR_DEV_MAXTIME`	Timeout, bevor der Lesevorgang beendet ist.

Weitere Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `ReadStr`.

Beispiel 1:

```
text := ReadStr(infile);
IF text = EOF THEN
  TPWrite "The file is empty";
  ...
```

Vor dem Lesen der Zeichenfolge aus der Datei wird geprüft, ob die Datei leer ist.

Beispiel 2:

Eine Datei enthält Folgendes:

```
<LF><SPACE><TAB>Hello<SPACE><SPACE>World<CR><LF>
text := ReadStr(infile);
```

`text` wird eine leere Zeichenfolge sein: Das erste Zeichen in der Datei, der standardmäßige Begrenzer `<LF>`.

```
text := ReadStr(infile\DiscardHeaders);
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.154 ReadStr - Liest eine Zeichenfolge aus einer Datei oder einem E/A-Gerät
RobotWare - OS
Fortsetzung

„text“ enthält <SPACE><TAB>Hello<SPACE><SPACE>World<CR>: Das erste Zeichen in der Datei, der Standardbegrenzer <LF>, wird ignoriert.

```
text := ReadStr(infile\RemoveCR\DiscardHeaders);
```

text enthält <SPACE><TAB>Hello<SPACE><SPACE>World:. Das erste Zeichen in der Datei, der Standardbegrenzer <LF>, wird ignoriert; das abschließende Wagenrücklaufzeichen wird entfernt

```
text := ReadStr(infile\Delim:=" \09"\RemoveCR\DiscardHeaders);
```

„text“ enthält "Hello": Die ersten Zeichen in der Datei, die dem Standardbegrenzer <LF> oder den in \Delim definierten Zeichen (Leerzeichen und Tabulator) entsprechen, werden ignoriert. Die Daten werden dann bis zum ersten Begrenzer übertragen, der aus der Datei gelesen, aber nicht in die Zeichenfolge übernommen wird. Ein neuer Aufruf derselben Anweisung gibt "World" zurück.

Beispiel 3

Eine Datei enthält Folgendes:

```
<CR><LF>Hello<CR><LF>
text := ReadStr(infile);
```

text enthält das <CR> (\0d) Zeichen: <CR> und die Zeichen<LF> werden aus der Datei ausgelesen, allerdings wird nur <CR> an den String Übergabe. Bei erneutem Aufruf derselben Anweisung wird "Hello\0d" zurückgegeben.

```
text := ReadStr(infile\RemoveCR);
```

text enthält eine leere Zeichenfolge: Die Zeichen <CR> und <LF> werden aus der Datei gelesen. <CR> wird übertragen, doch aus der Zeichenfolge entfernt. Ein neuer Aufruf derselben Anweisung gibt "Hello" zurück.

```
text := ReadStr(infile\Delim:="\0d");
```

text wird eine leere Zeichenfolge enthalten: <CR> wird aus der Datei gelesen, aber nicht an die Rückgabe-Zeichenfolge übertragen. Ein neuer Aufruf derselben Anweisung gibt erneut eine leere Zeichenfolge zurück. <LF> wird aus der Datei gelesen, aber nicht an die Rückgabe-Zeichenfolge übertragen.

```
text := ReadStr(infile\Delim:="\0d"\DiscardHeaders);
```

text enthält "Hello". Ein neuer Aufruf derselben Anweisung gibt "EOF" (Dateiende) zurück.

Einschränkungen

Diese Funktion kann nur für Dateien oder E/A-Geräte verwendet werden, die zum Lesen in einem zeichenorientierten Modus geöffnet wurden.

Syntax

```
ReadStr '('
  [IODevice ':='] <variable (VAR) of iodev>
  ['\' Delim ':='<expression (IN) of string>]
  ['\' RemoveCR]
  ['\' DiscardHeaders]
  ['\' Time ':=' <expression (IN) of num>]
  ['\' Line ':=' <expression (IN) of num>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs string.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Dateien, E/A-Geräte öffnen etc.	*Technisches Referenzhandbuch - RAPID Overview*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.155 ReadStrBin - Liest Zeichenfolge von binären E/A-Gerät oder einer Datei

Verwendung

`ReadStrBin` (*Read String Binary*) wird verwendet, um eine Zeichenfolge aus einem binären E/A-Gerät oder einer Datei zu lesen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `ReadStrBin`.

Beispiel 1:

```
VAR iodev file1;
VAR string text;
...
Open "HOME:", \File:= "FILE1.DOC", file1 \Read \Bin;
text := ReadStrBin (file1, 10);
IF text = EOF THEN
```

Der Variablen `text` wird eine `10`-stellige Zeichenfolge zugewiesen, die aus der Datei `file1` ausgelesen wird.

Vor dem Lesen der Zeichenfolge aus der Datei wird geprüft, ob die Datei leer ist.

Rückgabewert

Datentyp: `string`

Die Zeichenfolge wird aus dem angegebenen E/A-Gerät oder der Datei gelesen. Wenn die Datei leer ist (Dateiende), wird die Zeichenfolge `"EOF"` zurückgegeben.

Argumente

```
ReadStrBin (IODevice NoOfChars [\Time])
```

`IODevice`

Datentyp: `iodev`

Der Name (Verweis) des binären E/A-Geräts oder der Binärdatei, die gelesen werden soll.

`NoOfChars`

Number of Characters

Datentyp: `num`

Die Anzahl der Zeichen, die aus dem binären E/A-Gerät oder der Binärdatei gelesen werden soll.

`[\Time]`

Datentyp: `num`

Die maximale Dauer für den Lesevorgang (Timeout) in Sekunden. Wenn dieses Argument nicht angegeben wird, beträgt die maximale Dauer 60 Sekunden. Verwenden Sie für eine unendliche Wartezeit die vordefinierte Konstante `WAIT_MAX`.

Wenn diese Zeit abläuft, bevor der Lesevorgang abgeschlossen ist, wird die Fehlerbehandlungsroutine mit dem Fehlercode `ERR_DEV_MAXTIME` aufgerufen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn keine Fehlerbehandlungsroutine vorhanden ist, wird die Abarbeitung gestoppt.

Die Timeout-Funktion ist auch während des Programmstopps in Gebrauch und wird vom RAPID-Programm beim Programmstart berücksichtigt.

Programmabarbeitung

Die Funktion liest die angegebene Anzahl an Zeichen aus dem binären E/A-Gerät oder der Binärdatei.

Bei einem Neustart nach einem Stromausfall wird jede geöffnete Datei und jedes geöffnete E/A-Gerät im System geschlossen, und der E/A-Deskriptor in der Variablen vom Typ `iodev` wird zurückgesetzt.

Vordefinierte Daten

Mit der Konstanten `EOF` kann geprüft werden, ob die Datei beim Leseversuch leer war. Mit ihr kann auch der Lesevorgang am Ende der Datei gestoppt werden.

```
CONST string EOF := "EOF";
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_FILEACC`	Fehler beim Lesen.
`ERR_DEV_MAXTIME`	Timeout, bevor der Lesevorgang beendet ist.

Einschränkungen

Diese Funktion kann nur für E/A-Geräte oder Dateien verwendet werden, die zum Lesen in einem binären Modus geöffnet wurden.

Syntax

```
ReadStrBin '('
  [IODevice ':='] <variable (VAR) of iodev> ','
  [NoOfChars ':='] <expression (IN) of num>
  ['\' Time ':=' <expression (IN) of num>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Öffnen usw. von E/A-Geräten oder Dateien	*Technisches Referenzhandbuch - RAPID Overview*
Schreiben von binären Zeichenfolgen	*WriteStrBin - Schreibt einer Zeichenfolge auf ein binäres E/A-Gerät auf Seite 1130*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.156 ReadVar - Lesen einer Variablen von einem Gerät

Verwendung

`ReadVar` wird für das Lesen einer Variablen von einem Gerät verwendet, das mit der seriellen Sensorschnittstelle verbunden ist.

Konfigurationsbeispiel

Dies ist ein Beispiel für eine Sensorkanalkonfiguration.

Diese Parameter gehören zum Typ *Transmission Protocol* in der Parametergruppe *Communication*.

Name	Type	Remote Address	Remote Port
sen1:	SOCKDEV	192.168.125.101	6344

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ReadVar`.

Beispiel 1:

```
CONST num XCoord := 8;
CONST num YCoord := 9;
CONST num ZCoord := 10;

VAR pos SensorPos;

! Connect to the sensor device "sen1:" (defined in sio.cfg)
SenDevice "sen1:";

! Read a cartesian position from the sensor.

SensorPos.x := ReadVar ("sen1:", XCoord);
SensorPos.y := ReadVar ("sen1:", YCoord);
SensorPos.z := ReadVar ("sen1:", ZCoord);
```

Argumente

```
ReadVar (device, VarNo, [\TaskName])
```

`device`

Datentyp: `string`

Die E/A-Konfiguration für den verwendeten Sensor ist in der Datei „sio.cfg“ definiert.

`VarNo`

Datentyp: `num`

Mit dem Argument `VarNo` wird die zu lesende Variable gewählt.

`[ \TaskName ]`

Datentyp: `string`

Das Argument `TaskName` ermöglicht, auf Geräte in anderen RAPID-Tasks zuzugreifen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
SEN_NO_MEAS	Messfehler
SEN_NOREADY	Sensor kann Befehl nicht handhaben
SEN_GENERRO	Allgemeiner Sensorfehler
SEN_BUSY	Sensor belegt
SEN_UNKNOWN	Unbekannter Sensor
SEN_EXALARM	Externer Sensorfehler
SEN_CAALARM	Interner Sensorfehler
SEN_TEMP	Sensortemperaturfehler
SEN_VALUE	Unzulässiger Kommunikationswert
SEN_CAMCHECK	Sensorprüffehler
SEN_TIMEOUT	Kommunikationsfehler

Syntax

```
ReadVar '('
  [device ':='] <expression(IN) of string> ','
  [VarNo ':='] <expression (IN) of num> ','
  ['\' TaskName ':=' <expression (IN) of string> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Verbinden mit einer Sensoreinheit	*SenDevice - Verbinden mit einer Sensoreinheit auf Seite 693*
Schreiben einer Sensorvariablen	*WriteVar – Variable schreiben auf Seite 1132*
Schreiben eines Sensordatenblocks	*WriteBlock – Datenblock auf Gerät schreiben auf Seite 1121*
Lesen eines Sensordatenblocks	*ReadBlock - Lesen eines Datenblocks von einem Gerät auf Seite 584*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.157 RelTool - Verschiebung relativ zum Werkzeug

Verwendung

`RelTool` (*Relative Tool*) wird verwendet, um eine Verschiebung und/oder Rotation im aktiven Werkzeug-Koordinatensystem an einer Roboterposition auszuführen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `RelTool`.

Beispiel 1:

```
MoveL RelTool (p1, 0, 0, 100), v100, fine, tool1;
```

Der Roboter wird an eine Position bewegt, die sich `100` mm von `p1` in der z-Richtung des Werkzeugs befindet.

Beispiel 2:

```
MoveL RelTool (p1, 0, 0, 0 \Rz:= 25), v100, fine, tool1;
```

Das Werkzeug wird 25° um seine z-Achse gedreht.

Rückgabewert

Datentyp: `robtarget`

Die neue Position mit hinzugefügter Verschiebung und/oder Rotation (falls vorhanden) in Relation zum aktiven Werkzeug.

Argumente

```
RelTool (Point Dx Dy Dz [\Rx] [\Ry] [\Rz])
```

Hinweis

Die Rotationen werden in der folgenden Reihenfolge ausgeführt, wenn zwei oder drei Rotationen gleichzeitig angegeben werden:

1. Rotation um die x-Achse
2. Rotation um die neue y-Achse
3. Rotation um die neue z-Achse

`Point`

Datentyp: `robtarget`

Die eingegebene Roboterposition. Der Orientierungsteil dieser Position definiert die aktuelle Orientierung des Werkzeug-Koordinatensystems.

`Dx`

Datentyp: `num`

Die Verschiebung in mm in die x-Richtung des Werkzeug-Koordinatensystems.

`Dy`

Datentyp: `num`

Die Verschiebung in mm in die y-Richtung des Werkzeug-Koordinatensystems.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Dz`

Datentyp: `num`

Die Verschiebung in mm in die z-Richtung des Werkzeug-Koordinatensystems.

`[\Rx]`

Datentyp: `num`

Die Rotation in Grad um die x-Achse des Werkzeug-Koordinatensystems.

`[\Ry]`

Datentyp: `num`

Die Rotation in Grad um die y-Achse des Werkzeug-Koordinatensystems.

`[\Rz]`

Datentyp: `num`

Die Rotation in Grad um die z-Achse des Werkzeug-Koordinatensystems.

Syntax

```
RelTool '('
  [ Point ':=' ] < expression (IN) of robtarget> ','
  [Dx ':='] <expression (IN) of num> ','
  [Dy ':='] <expression (IN) of num> ','
  [Dz ':='] <expression (IN) of num>
  ['\' Rx ':=' <expression (IN) of num> ]
  ['\' Ry ':=' <expression (IN) of num> ]
  ['\' Rz ':=' <expression (IN) of num> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `robtarget`.

Weitere Informationen

Information über	Siehe
Positionsdaten	*robtarget - Positionsdaten auf Seite 1778*
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.158 RemainingRetries Restliche Anzahl an Wiederholungen

Verwendung

`RemainingRetries` wird verwendet, um die Anzahl der `RETRY` zu ermitteln, die in der Fehlerbehandlungsroutine des Programms noch ausgeführt werden können. Die maximale Anzahl der Wiederholungen ist in der Konfiguration definiert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `RemainingRetries`.

Beispiel 1:

```
...
ERROR
  IF RemainingRetries() > 0 THEN
    RETRY;
  ELSE
    TRYNEXT;
  ENDIF
...
```

Dieses Programm wiederholt die Instruktion trotz des Fehlers, bis die maximale Anzahl von Wiederholungen ausgeführt wurde, und versucht dann, die nächste Instruktion auszuführen.

Rückgabewert

Datentyp: `num`

Der Rückgabewert gibt an, wie viele Wiederholungen bis zur maximalen Anzahl noch ausgeführt werden können.

Syntax

```
RemainingRetries '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Fehlerbehandlungen	*Technisches Referenzhandbuch - RAPID Overview*
Wiederaufnahme der Abarbeitung nach einem Fehler	*RETRY - Wiederaufnahme der Abarbeitung nach einem Fehler auf Seite 622*
Konfigurieren der maximalen Anzahl an Wiederholversuchen	*Technisches Referenzhandbuch - Systemparameter*, Abschnitt *General RAPID*
Zurücksetzen der Anzahl an gezählten Wiederholungen	*ResetRetryCount - Zurücksetzen der Anzahl an Wiederholungen auf Seite 618*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.159 RMQGetSlotName - Abrufen des Namens eines RMQ-Clients

Verwendung

`RMQGetSlotName` (*RAPID Mesasage Queue Get Slot Name*) wird zum Abrufen des Slot-Namens von RMQ oder eines SDK-Clients aus einer angegebenen Slot-Identität, d. h. aus einem angegeben `rmqslot`, verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `RMQGetSlotName`.

Beispiel 1:

```
VAR rmqslot slot;
VAR string client_name;
RMQFindSlot slot, "RMQ_T_ROB1";
...
client_name := RMQGetSlotName(slot);
TPWrite "Name of the client: " + client_name;
```

In dem Beispiel wird veranschaulicht, wie mit der Identität eines Clients der Name des Clients abgerufen wird.

Rückgabewert

Datentyp: `string`

Der Name des zurückgegebenen Clients. Dies kann ein RMQ-Name oder der Name eines Robot Application Builder-Clients sein, der die RMQ-Funktionalität verwendet.

Argumente

```
RMQGetSlotName (Slot)
```

`Slot`

Datentyp: `rmqslot`

Die Slot-Identitätsnummer des Clients, dessen Name gesucht werden soll.

Programmabarbeitung

Mit der Instruktion `RMQGetSlotName` wird der Name des Clients mit der im Argument `Slot` angegebenen Identitätsnummer gesucht. Der Client kann ein anderer RMQ oder ein SDK-Client sein.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_RMQ_INVALID`	Der Ziel-Slot wurde nicht verbunden oder ist nicht mehr verfügbar. Wenn keine Verbindung besteht, muss `RMQFindSlot` aufgerufen werden. Wenn der Slot nicht verfügbar ist, wurde die Verbindung eines Remote-Clients mit der Steuerung getrennt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.159 RMQGetSlotName - Abrufen des Namens eines RMQ-Clients

FlexPendant Interface, PC Interface, or Multitasking

Fortsetzung

Syntax

```
RMQGetSlotName '('
  [ Slot ':=' ] < variable (VAR) of rmqslot > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Beschreibung der Funktion von RAPID Message Queue	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer eines RAPID Message Queue-Task oder eines SDK-Clients	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Sendet Daten an die Warteschlange für einen RAPID-Task oder einen SDK-Client.	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Abrufen der ersten Meldung von RAPID Message Queue.	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange eines RAPID-Task oder eines SDK-Clients und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
`RMQ Slot`	*rmqslot - Identitätsnummer eines RMQ-Clients auf Seite 1776*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.160 RobName - Abrufen des TCP-Roboternamens

Verwendung

`RobName` (*Robot Name*) wird verwendet, um den Namen des TCP-Roboters in einer Programmtask abzurufen. Wenn kein TCP-Roboter von dieser Task gesteuert wird, gibt die Funktion eine leere Zeichenfolge zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `RobName`.

Siehe auch *Weitere Beispiele auf Seite 1481*.

Beispiel 1:

```
VAR string my_robot;
...
my_robot := RobName();
IF my_robot="" THEN
  TPWrite "This task does not control any TCP robot";
ELSE
  TPWrite "This task controls TCP robot with name "+ my_robot;
ENDIF
```

Der Name des TCP-Roboters, der von dieser Programmtask gesteuert wird, wird an das FlexPendant geschrieben. Wenn kein TCP-Roboter gesteuert wird, wird geschrieben, dass die Task keinen Roboter steuert.

Rückgabewert

Datentyp: `string`

Der Name der mechanischen Einheit für den TCP-Roboter, der von dieser Programmtask gesteuert wird. Es wird eine leere Zeichenfolge zurückgegeben, wenn kein TCP-Roboter gesteuert wird.

Weitere Beispiele

Weitere Beispiele zur Verwendung der Instruktion `RobName` sind unten aufgeführt.

Beispiel 1:

```
VAR string my_robot;
...
IF TaskRunRob() THEN
   my_robot := RobName();
   TPWrite "This task controls robot with name "+ my_robot;
ENDIF
```

Wenn diese Programmtask einen TCP-Roboter steuert, wird der Name dieses TCP-Roboters an das FlexPendant geschrieben.

Syntax

```
   RobName '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.160 RobName - Abrufen des TCP-Roboternamens

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Prüfen, ob die Task einen TCP-Roboter steuert	*TaskRunRob - Überprüfen, ob die Task einen Roboter steuert auf Seite 1542*
Prüfen, ob die Task eine mechanische Einheit ausführt	*TaskRunMec - Überprüfen, ob die Task eine mechanische Einheit steuert auf Seite 1541*
Abrufen des Namens von mechanischen Einheiten im System	*GetNextMechUnit – Abrufen des Namens und der Daten für mechanische Einheiten auf Seite 1325*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Instruktionen, Funktionen und Datentypen*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.161 RobOS - Prüfen, ob die Abarbeitung auf RC oder VC erfolgt

Verwendung

`RobOS` (*Robot Operating System*) kann verwendet werden, um zu überprüfen, ob die Abarbeitung am Robot Controller RC oder Virtual Controller VC erfolgt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `RobOS`.

Beispiel 1:

```
IF RobOS() THEN
  ! Execution statements in RC
ELSE
  ! Execution statements in VC
ENDIF
```

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn die Abarbeitung an der Robotersteuerung RC abläuft, sonst `FALSE`.

Syntax

```
RobOS '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.162 Round - Rundet einen numerischen Wert

Verwendung

`Round` wird verwendet, um einen numerischen Wert auf eine angegebene Anzahl von Dezimalstellen oder eine Ganzzahl zu runden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `Round`.

Beispiel 1:

```
VAR num val;
val := Round(0.3852138\Dec:=3);
```

Die Variable `val` erhält den Wert `0.385`.

Beispiel 2:

```
val := Round(0.3852138\Dec:=1);
```

Die Variable `val` erhält den Wert `0.4`.

Beispiel 3

```
val := Round(0.3852138);
```

Die Variable `val` erhält den Wert `0`.

Beispiel 4:

```
val := Round(0.3852138\Dec:=6);
```

Die Variable `val` erhält den Wert `0.385214`.

Rückgabewert

Datentyp: `num`

Der numerische Wert, auf die angegebene Anzahl an Dezimalstellen gerundet.

Argumente

```
Round ( Val [\Dec])
```

`Val`

Value

Datentyp: `num`

Der zu rundende numerische Wert.

`[\Dec]`

Decimals

Datentyp: `num`

Anzahl an Dezimalstellen.

Wenn die angegebene Anzahl an Dezimalstellen 0 ist oder das Argument weggelassen wird, wird der Wert auf eine Ganzzahl gerundet.

Die Anzahl an Dezimalstellen darf nicht als negativer Wert oder mit größerer Genauigkeit als die verfügbare Genauigkeit für numerische Werte angegeben werden.

Maximale Anzahl an verwendbaren Dezimalzahlen ist 6.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
Round '('
  [ Val ':=' ] <expression (IN) of num>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Abschneiden eines Werts	*Trunc - Schneidet einen numerischen Wert ab auf Seite 1565*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.163 RoundDnum - Rundet einen numerischen Wert

Verwendung

`RoundDnum` wird verwendet, um einen numerischen Wert auf eine angegebene Anzahl von Dezimalstellen oder eine Ganzzahl zu runden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `RoundDnum`.

Beispiel 1:

```
VAR dnum val;
val := RoundDnum(0.3852138754655357\Dec:=3);
```

Die Variable `val` erhält den Wert `0.385`.

Beispiel 2:

```
val := RoundDnum(0.3852138754655357\Dec:=1);
```

Die Variable `val` erhält den Wert `0.4`.

Beispiel 3

```
val := RoundDnum(0.3852138754655357);
```

Die Variable `val` erhält den Wert `0`.

Beispiel 4:

```
val := RoundDnum(0.3852138754655357\Dec:=15);
```

Die Variable `val` erhält den Wert `0.385213875465536`.

Beispiel 5:

```
val := RoundDnum(1000.3852138754655357\Dec:=15);
```

Die Variable `val` erhält den Wert `1000.38521387547`.

Rückgabewert

Datentyp: `dnum`

Der numerische Wert, auf die angegebene Anzahl an Dezimalstellen gerundet.

Argumente

```
RoundDnum ( Val [\Dec])
```

`Val`

Value

Datentyp: `dnum`

Der zu rundende numerische Wert.

`[\Dec]`

Decimals

Datentyp: `num`

Anzahl an Dezimalstellen.

Wenn die angegebene Anzahl an Dezimalstellen 0 ist oder das Argument weggelassen wird, wird der Wert auf eine Ganzzahl gerundet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Anzahl an Dezimalstellen darf nicht als negativer Wert oder mit größerer Genauigkeit als die verfügbare Genauigkeit für numerische Werte angegeben werden.

Maximale Anzahl an verwendbaren Dezimalzahlen ist 15.

Syntax

```
RoundDnum '('
  [ Val ':=' ] <expression (IN) of dnum>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Runden eines Werts	*Round - Rundet einen numerischen Wert auf Seite 1484*
Abschneiden eines Werts	*Trunc - Schneidet einen numerischen Wert ab auf Seite 1565*
Abschneiden eines Werts	*TruncDnum - Schneidet einen numerischen Wert ab auf Seite 1567*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.164 RunMode - Lesen des Abarbeitungsmodus

Verwendung

`RunMode` (*Running Mode*) wird verwendet, um den aktuellen Modus der Programmtask zu lesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `RunMode`.

Beispiel 1:

```
IF RunMode() = RUN_CONT_CYCLE THEN
...
ENDIF
```

Der Programmabschnitt wird nur für den kontinuierlichen oder zyklischen Modus abgearbeitet.

Rückgabewert

Datentyp: `symnum`

Der aktuelle Abarbeitungsmodus wird definiert wie in nachstehender Tabelle beschrieben.

Rückgabewert	Symbolische Konstante	Kommentar
0	`RUN_UNDEF`	Undefinierter Abarbeitungsmodus
1	`RUN_CONT_CYCLE`	Kontinuierlicher oder zyklischer Abarbeitungsmodus
2	`RUN_INSTR_FWD`	Vorwärtsabarbeitungsmodus für Instruktion
3	`RUN_INSTR_BWD`	Rückwärtsabarbeitungsmodus für Instruktion
4	`RUN_SIM`	Simulierter Abarbeitungsmodus. Noch nicht veröffentlicht.
5	`RUN_STEP_MOVE`	Bewegungsinstruktionen in Vorwärtsabarbeitung und logische Instruktionen in kontinuierlichem Abarbeitungsmodus

Argumente

```
RunMode ( [ \Main] )
```

`[ \Main ]`

Datentyp: `switch`

Gibt den aktuellen Modus der Task zurück, wenn es sich um eine Bewegungstask handelt. Bei Verwendung in einer Nichtbewegungstask wird der aktuelle Modus der Bewegungstask wiedergegeben, mit der die Nichtbewegungstask verbunden ist.

Wenn dieses Argument weggelassen wird, spiegelt der Rückgabewert stets den aktuellen Abarbeitungsmodus für die Programmtask, die die Funktion `RunMode` abarbeitet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Syntax

```
RunMode '('
  ['\' Main] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `symnum`.

Weitere Informationen

Information über	Siehe
Lesen des Betriebsmodus	*OpMode - Lesen des Betriebsmodus auf Seite 1414*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.165 SafetyControllerGetChecksum - Abrufen der Prüfsumme der Benutzerkonfigurationsdatei

Verwendung

`SafetyControllerGetChecksum` wird verwendet, um die Prüfsumme der Sicherheitssteuerung für die Benutzerkonfigurationsdatei abzurufen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `SafetyControllerGetChecksum`.

Beispiel 1:

```
VAR string mystring;
...
mystring:=SafetyControllerGetChecksum();
```

Abrufen der Prüfsumme für die Benutzerkonfigurationsdatei und Speichern in der Variablen `mystring`.

Rückgabewert

Datentyp: `string`

Die Prüfsumme für die Benutzerkonfiguration.

Syntax

```
SafetyControllerGetChecksum '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
`SafetyControllerGetOpModePinCode`	*SafetyControllerGetOpModePinCode - Pin-Code der Betriebsart abrufen auf Seite 1491*
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Prüfsumme für geschützte Parameter abrufen auf Seite 1493*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung auf Seite 1492*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Initiierung des Hardware-Synchronisationsverfahrens auf Seite 652*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.166 SafetyControllerGetOpModePinCode - Pin-Code der Betriebsart abrufen

Verwendung

`SafetyControllerGetOpModePinCode` wird verwendet, um den Pin-Code des Betriebsmodus für den Keyless Mode Selector zu erhalten.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `SafetyControllerGetOpModePinCode`.

Beispiel 1:

```
VAR string mystring;
...
mystring:=SafetyControllerGetOpModePinCode();
```

Rufen Sie den Pin-Code der Betriebsart für den Keyless Mode Selector und speichern sie ihn in der Variablen `mystring`.

Rückgabewert

Datentyp: `string`

Der Pin-Code für den Keyless Mode Selector.

Syntax

```
SafetyControllerGetOpModePinCode '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
`SafetyControllerGetChecksum`	*SafetyControllerGetChecksum - Abrufen der Prüfsumme der Benutzerkonfigurationsdatei auf Seite 1490*
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Prüfsumme für geschützte Parameter abrufen auf Seite 1493*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung auf Seite 1492*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Initiierung des Hardware-Synchronisationsverfahrens auf Seite 652*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.167 SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung

Verwendung

`SafetyControllerGetSWVersion` wird verwendet, um die Firmware-Version der Sicherheitssteuerung abzurufen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `SafetyControllerGetSWVersion`.

Beispiel 1:

```
VAR string mystring;
...
mystring:=SafetyControllerGetSWVersion();
```

Firmware-Version der Sicherheitssteuerung abrufen und Speichern in der Variablen `mystring`.

Rückgabewert

Datentyp: `string`

Die Firmware-Version der Sicherheitssteuerung. Eine Zeichenkette mit "VC" wird ausgegeben, wenn diese Funktion vom Virtual Controller verwendet wird.

Syntax

```
SafetyControllerGetSWVersion '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
`SafetyControllerGetOpModePinCode`	*SafetyControllerGetOpModePinCode - Pin-Code der Betriebsart abrufen auf Seite 1491*
`SafetyControllerGetUserChecksum`	*SafetyControllerGetUserChecksum - Prüfsumme für geschützte Parameter abrufen auf Seite 1493*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung auf Seite 1492*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Initiierung des Hardware-Synchronisationsverfahrens auf Seite 652*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.168 SafetyControllerGetUserChecksum - Prüfsumme für geschützte Parameter abrufen

Verwendung

`SafetyControllerGetUserChecksum` wird verwendet, um die Prüfsumme der Sicherheitssteuerung für den Bereich der geschützten Parameter in der Benutzerkonfigurationsdatei abzurufen.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `SafetyControllerGetUserChecksum`.

Beispiel 1:

```
VAR string mystring;
...
mystring:=SafetyControllerGetUserChecksum();
```

Abrufen der Prüfsumme für den Bereich der geschützten Parameter der Benutzerkonfigurationsdatei und Speichern in der Variablen `mystring`.

Rückgabewert

Datentyp: `string`

Prüfsumme für den Bereich der geschützten Parameter.

Syntax

```
SafetyControllerGetUserChecksum '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
`SafetyControllerGetOpModePinCode`	*SafetyControllerGetOpModePinCode - Pin-Code der Betriebsart abrufen auf Seite 1491*
`SafetyControllerGetChecksum`	*SafetyControllerGetChecksum - Abrufen der Prüfsumme der Benutzerkonfigurationsdatei auf Seite 1490*
`SafetyControllerGetSWVersion`	*SafetyControllerGetSWVersion - Abrufen der Firmware-Version der Sicherheitssteuerung auf Seite 1492*
`SafetyControllerSyncRequest`	*SafetyControllerSyncRequest - Initiierung des Hardware-Synchronisationsverfahrens auf Seite 652*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.169 Sin - Berechnet den Sinuswert.

Verwendung

`Sin`(*Sine*) wird zur Berechnung des Sinuswerts aus einem Winkel verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Sin`.

Beispiel 1:

```
VAR num angle;
VAR num value;
...
...
value := Sin(angle);
```

`value` ruft den Sinuswert von `angle` ab.

Rückgabewert

Datentyp: `num`

Der Sinuswert, Bereich [-1, 1].

Argumente

```
Sin (Angle)
```

`Angle`

Datentyp: `num`

Der Wert des Winkels in Grad.

Syntax

```
Sin '('
  [Angle ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.170 SinDnum - Berechnet den Sinuswert

Verwendung

(SinDnum)*Sine dnum* wird zur Berechnung des Sinuswerts aus einem Winkel für die Datentypen dnum verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion SinDnum.

Beispiel 1:

```
VAR dnum angle;
VAR dnum value;
...
...
value := SinDnum(angle);
```

value ruft den Sinuswert von angle ab.

Rückgabewert

Datentyp: dnum

Der Sinuswert, Bereich [-1, 1].

Argumente

```
SinDnum (Angle)
```

Angle

Datentyp: dnum

Der Wert des Winkels in Grad.

Syntax

```
SinDnum '('
  [Angle ':='] <expression (IN) of dnum> ')'
```

Eine Funktion mit einem Rückgabewert des Typs dnum.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.171 SocketGetStatus - Abrufen des aktuellen Socket-Status

Verwendung

`SocketGetStatus` gibt den aktuellen Status eines Sockets wieder.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `SocketGetStatus`.

Siehe auch *Weitere Beispiele auf Seite 1496*.

Beispiel 1:

```
VAR socketdev socket1;
VAR socketstatus state;
...
SocketCreate socket1;
state := SocketGetStatus( socket1 );
```

Der Socket-Status `SOCKET_CREATED` wird in der Variablen `state` gespeichert.

Rückgabewert

Datentyp: `socketstatus`

Der aktuelle Status des Sockets.

Nur die vordefinierten, symbolischen Konstanten des Typs `socketstatus` können zur Überprüfung des Status verwendet werden.

Argumente

```
SocketGetStatus( Socket )
```

`Socket`

Datentyp: `socketdev`

Die Socket-Variable, deren Status erfasst werden soll.

Programmabarbeitung

Die Funktion gibt einen der folgenden vordefinierten Socket-Status zurück:

`SOCKET_CREATED`, `SOCKET_CONNECTED`, `SOCKET_BOUND`, `SOCKET_LISTENING` oder `SOCKET_CLOSED`.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `SocketGetStatus`.

Beispiel 1:

```
VAR socketstatus status;
VAR socketdev my_socket;
...
SocketCreate my_socket;
SocketConnect my_socket, "192.168.0.1", 1025;
! A lot of RAPID code
status := SocketGetStatus( my_socket );
!Check which instruction that was executed last, not the state of
!the socket
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
IF status = SOCKET_CREATED THEN
  TPWrite "Instruction SocketCreate has been executed";
ELSEIF status = SOCKET_CLOSED THEN
  TPWrite "Instruction SocketClose has been executed";
ELSEIF status = SOCKET_BOUND THEN
  TPWrite "Instruction SocketBind has been executed";
ELSEIF status = SOCKET_LISTENING THEN
  TPWrite "Instruction SocketListen or SocketAccept has been
       executed";
ELSEIF status = SOCKET_CONNECTED THEN
  TPWrite "Instruction SocketConnect, SocketReceive or SocketSend
       has been executed";
ELSE
  TPWrite "Unknown socket status";
ENDIF
```

Es wird ein Client-Socket erstellt und mit einem Remote-Computer verbunden. Bevor der Socket in einer `SocketSend`-Instruktion verwendet wird, wird der Status des Sockets überprüft, um festzustellen, ob es noch verbunden ist.

Einschränkungen

Der Status einer Buchse kann nur durch die Abarbeitung einer RAPID-Buchseninstruktion geändert werden. Wenn die Buchse z. B. verbunden ist und diese Verbindung zu einem späteren Zeitpunkt unterbrochen wird, wird dies nicht durch die `SocketGetStatus`-Funktion gemeldet. Stattdessen wird ein Fehler zurückgegeben, wenn die Buchse in einer `SocketSend`- oder `SocketReceive`-Instruktion verwendet wird.

Syntax

```
SocketGetStatus '('
  [Socket ':='] <variable (VAR) of socketdev>')'
```

Eine Funktion mit einem Rückgabewert des Typs `socketstatus`.

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.171 SocketGetStatus - Abrufen des aktuellen Socket-Status

Socket Messaging

Fortsetzung

Information über	Siehe
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.172 SocketPeek – Überprüfung eines Sockets auf vorhandene Daten hin

Verwendung

`SocketPeek` wird verwendet, um zu überprüfen, ob sich auf einem Socket Daten befinden. Es sendet die Bytezahl zurück, die von einem bestimmten Socket empfangen werden kann.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `SocketPeek`.

Beispiel 1:

```
VAR socketdev socket1;
VAR socketdev client_socket;
VAR num peek_value;
...
SocketCreate socket1;
SocketBind socket1, "192.168.0.1", 1025;
SocketListen socket1;
SocketAccept socket1, client_socket;
..
peek_value := SocketPeek( client_socket );
IF peek_value >= 64 THEN
  SocketReceive client_socket \Str := str_data \ReadNoOfBytes:=64;
  ..
ELSE
  ! Not enough data to receive. Do something else.
ENDIF
```

Zunächst wird ein Server-Socket erstellt und an Port 1025 im Steuerungsnetzwerk mit der Adresse 192.168.0.1 gebunden. Dann wird `SocketPeek` benutzt, um zu überprüfen, ob auf dem Socket Daten mit einem Volumen von 64 Byte vorhanden sind, die empfangen werden können.

Rückgabewert

Datentyp: `num`

Die Anzahl an Bytes, die auf einem bestimmten Socket verfügbar ist.

Argumente

```
SocketPeek ( Socket )
```

`Socket`

Datentyp: `socketdev`

Die Socket-Variable, die überprüft werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.172 SocketPeek – Überprüfung eines Sockets auf vorhandene Daten hin

Socket Messaging

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_SOCK_CLOSED	Der Socket ist geschlossen. Verbindung ist unterbrochen.
ERR_SOCK_NOT_BOUND	Der Socket wurde nicht an eine Adresse gebunden. Bei der Verwendung von Typ Datagramm-Protokoll UDP/IP.
ERR_SOCK_NOT_CONN	Der Socket ist nicht verbunden.

Einschränkungen

Nach einem Neustart infolge eines Netzausfalls sind alle Sockets geschlossen. Dieses Problem kann mithilfe der Fehlerbehandlung behoben werden.

Die maximale Datenmenge, die innerhalb eines Aufrufs empfangen werden kann, ist auf 1024 Byte beschränkt. Aus diesem Grund kann SocketPeek maximal 1024 zurücksenden.

Syntax

```
SocketPeek '('
  [Socket ':='] <variable (VAR) of socketdev>')'
```

Eine Funktion mit einem Rückgabewert des Typs num.

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Herstellen einer Verbindung zum Remote-Computer (nur Client)	*SocketConnect - Herstellen einer Verbindung zu einem Remote-Computer auf Seite 751*
Senden von Daten an Remote-Computer	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*
Senden von Daten an Remote-Computer	*SocketSendTo - Daten an Remote-Computer senden auf Seite 774*
Schließen des Sockets	*SocketClose - Schließen eines Sockets auf Seite 749*
Binden eines Sockets (nur Server)	*SocketBind - Binden eines Sockets an meine IP-Adresse und meinen Port auf Seite 746*
Prüfen auf Verbindungen (nur Server)	*SocketListen - Prüfen auf eingehende Verbindungen auf Seite 757*
Akzeptieren von Verbindungen (nur Server)	*SocketAccept - Akzeptieren einer eingehenden Verbindung auf Seite 742*
Abrufen des aktuellen Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Beispiel für eine Client-Socketanwendung	*SocketSend - Senden von Daten an Remote-Computer auf Seite 769*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.172 SocketPeek – Überprüfung eines Sockets auf vorhandene Daten hin

Socket Messaging

Fortsetzung

Information über	Siehe
Daten vom Remote-Computer empfangen	*SocketReceive - Empfangen von Daten vom Remote-Computer auf Seite 759*
Daten vom Remote-Computer empfangen	*SocketReceiveFrom - Daten vom Remote-Computer empfangen auf Seite 764*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.173 Sqrt - Berechnen der Quadratwurzel

Verwendung

`Sqrt` (*Square root*) wird zur Berechnung der Quadratwurzel verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Sqrt`.

Beispiel 1:

```
VAR num x_value;
VAR num y_value;
...
...
y_value := Sqrt( x_value);
```

`y-value` ruft den Quadratwurzelwert von `x_value` ab, d. h., √(`x_value`).

Rückgabewert

Datentyp: `num`

Wert der Quadratwurzel (√).

Argumente

```
Sqrt (Value)
```

`Value`

Datentyp: `num`

Der Argumentwert der Quadratwurzel, d. h. √`value`.

`Value` muss ≥0 sein.

Einschränkungen

Die Abarbeitung der Funktion `Sqrt(x)` ergibt einen Fehler, wenn `x` < 0.

Syntax

```
Sqrt '('
  [Value ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Berechnung der Quadratwurzel eines numerischen dnum-Werts	*SqrtDnum - Berechnen der Quadratwurzel auf Seite 1503*
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.174 SqrtDnum - Berechnen der Quadratwurzel

Verwendung

`SqrtDnum` (*Square root dnum)* wird zur Berechnung der Quadratwurzel verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `SqrtDnum`.

Beispiel 1:

```
VAR dnum x_value;
VAR dnum y_value;
...
...
y_value := SqrtDnum(x_value);
```

`y_value` ruft den Quadratwurzelwert von `x_value` ab, d. h., √(`x_value`).

Rückgabewert

Datentyp: `dnum`

Wert der Quadratwurzel (√).

Argumente

```
SqrtDnum (Value)
```

`Value`

Datentyp: `dnum`

Der Argumentwert der Quadratwurzel, d. h. √`value`.

`Value` muss ≥0 sein.

Einschränkungen

Die Abarbeitung der Funktion `Sqrt(x)` ergibt einen Fehler, wenn `x` < 0.

Syntax

```
SqrtDnum '('
  [ Value ':=' ] < expression (IN) of dnum > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Berechnung der Quadratwurzel eines numerischen num-Werts	*Sqrt - Berechnen der Quadratwurzel auf Seite 1502*
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.175 STCalcForce - Berechnen der Elektrodenkraft für ein Servowerkzeug

Verwendung

`STCalcForce` wird zur Berechnung der Elektrodenkraft für ein Servowerkzeug verwendet. Diese Funktion wird beispielsweise zur Berechnung der maximal zulässigen Elektrodenkraft für ein Servowerkzeug verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `STCalcForce`.

Beispiel 1:

```
VAR num tip_force;
tip_force := STCalcForce(gun1, 7);
```

Berechnung der Elektrodenkraft bei einem gewünschten Motordrehmoment von `7` Nm.

Rückgabewert

Datentyp: `num`

Die berechnete Elektrodenkraft [N].

Argumente

```
STCalcForce ToolName MotorTorque
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`MotorTorque`

Datentyp: `num`

Das gewünschte Motordrehmoment [Nm].

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_SGUN`	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.

Syntax

```
STCalcForce '('
  [ ToolName ':=' ] < expression (IN) of string > ','
  [ MotorTorque ':=' ] < expression (IN) of num > ';'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Öffnen eines Servowerkzeugs	*STOpen - Öffnen eines Servowerkzeugs auf Seite 836*
Schließen eines Servowerkzeugs	*STClose - Schließen eines Servowerkzeugs auf Seite 817*
Berechnen des Motordrehmoments	*STCalcTorque - Berechnen des Motordrehmoments für ein Servowerkzeug auf Seite 1506*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.176 STCalcTorque - Berechnen des Motordrehmoments für ein Servowerkzeug

Verwendung

`STCalcTorque` wird zur Berechnung des Motordrehmoments für ein Servowerkzeug verwendet. Diese Funktion wird z. B. bei einer Kraftkalibrierung benutzt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `STCalcTorque`.

Beispiel 1:

```
VAR num curr_motortorque;
curr_motortorque := STCalcTorque( gun1, 1000);
```

Berechnung der Elektrodenkraft bei einem gewünschten Motordrehmoment von `1000` Nm.

Rückgabewert

Datentyp: `num`

Das berechnete Motordrehmoment [Nm].

Argumente

```
STCalcTorque ToolName TipForce
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`TipForce`

Datentyp: `num`

Die gewünschte Elektrodenkraft [N].

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_SGUN`	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.

Syntax

```
STCalcTorque '('
  [ ToolName ':=' ] < expression (IN) of string > ','
  [ TipForce ':=' ] < expression (IN) of num > ';'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Öffnen eines Servowerkzeugs	*STOpen - Öffnen eines Servowerkzeugs auf Seite 836*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.176 STCalcTorque - Berechnen des Motordrehmoments für ein Servowerkzeug

Servo tool control

Fortsetzung

Information über	Siehe
Schließen eines Servowerkzeugs	*STClose - Schließen eines Servowerkzeugs auf Seite 817*
Berechnen der Elektrodenkraft	*STCalcForce - Berechnen der Elektrodenkraft für ein Servowerkzeug auf Seite 1504*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.177 STIsCalib - Überprüfen, ob ein Servowerkzeug kalibriert ist

Verwendung

Mit `STIsCalib` wird überprüft, ob ein Servowerkzeug kalibriert ist, d. h. ob die Zangenelektroden kalibriert oder synchronisiert sind

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `STIsCalib`.

Beispiel 1:

```
IF STIsCalib(gun1\sguninit) THEN
  ...
ELSE
  !Start the gun calibration
  STCalib gun1\TipChg;
ENDIF
```

Beispiel 2:

```
IF STIsCalib(gun1\sgunsynch) THEN
  ...
ELSE
  !Start the gun calibration to synchronize the gun position with
       the revolution counter
  STCalib gun1\ToolChg;
ENDIF
```

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn das überprüfte Werkzeug kalibriert ist (der Abstand zwischen den Werkzeugelektroden ist kalibriert) oder synchronisiert ist (die Position der Werkzeugelektroden ist mit dem Umdrehungszähler des Werkzeugs synchronisiert).

`FALSE`, wenn das überprüfte Werkzeug nicht kalibriert oder synchronisiert ist.

Argumente

```
STIsCalib ToolName [\sguninit] | [\sgunsynch]
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`[ \sguninit ]`

Datentyp: `switch`

Mit diesem Argument wird überprüft, ob die Zangenposition initialisiert und kalibriert ist.

`[ \sgunsynch ]`

Datentyp: `switch`

Mit diesem Argument wird überprüft, ob die Zangenposition mit dem Umdrehungszähler synchronisiert ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.177 STIsCalib - Überprüfen, ob ein Servowerkzeug kalibriert ist

Servo Tool Control

Fortsetzung

Syntax

```
STIsCalib '('
  [ ToolName ':=' ] < expression (IN) of string >
  [ '\' sguninit ] | [ '\'sgunsynch ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Kalibrieren eines Servowerkzeugs	*STCalib - Kalibrieren eines Servowerkzeugs auf Seite 812*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.178 STIsClosed - Prüfen, ob ein Servowerkzeug geschlossen ist

Verwendung

`STIsClosed` wird verwendet, um zu prüfen, ob ein Servowerkzeug geschlossen ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `STIsClosed`.

Beispiel 1:

```
IF STIsClosed(gun1) THEN
  !Start the weld process
  Set weld_start;
ELSE
  ...
ENDIF
```

Es wird geprüft, ob die Zange geschlossen ist.

Beispiel 2:

```
STClose "sgun", 1000, 3 \Conc;
WHILE NOT(STIsClosed("sgun"\RetThickness:=thickness)) DO
  WaitTime 0.1;

ENDWHILE
IF thickness > max_thickness THEN...
```

Es wird begonnen, die Zange mit dem Namen `sgun` zu schließen. Die nächste Instruktion, in der das Programm darauf wartet, dass die Zange geschlossen wird, folgt sofort darauf. Der erreichte thickness-Wert wird abgelesen, wenn die Instruktion `STIsClosed` den Wert `TRUE` zurückgegeben hat.

Beispiel 3

Beispiele von unzulässigen Kombinationen:

```
STClose "sgun", 1000, 3 \RetThickness:=thickness \Conc;
WHILE NOT(STIsClosed("sgun"\RetThickness:=thickness_2)) DO;
...
```

Die Zange wird geschlossen. Der Parameter thickness enthält **keinen** gültigen Wert, weil der Schalter `\Conc` verwendet wird. Es wird darauf gewartet, dass die Zange geschlossen ist. Wenn die Zange geschlossen ist und `STIsClosed` den Wert `TRUE` zurückgibt, enthält der Parameter thickness_2 einen gültigen Wert, weil der Schalter `\Conc` für `STClose` verwendet wurde.

```
STClose "sgun", 1000, 3 \RetThickness:=thickness;
WHILE NOT(STIsClosed("sgun"\RetThickness:=thickness_2)) DO;
...
```

Die Zange wird geschlossen. Der Parameter thickness enthält einen gültigen Wert, wenn die Zange geschlossen wurde, da der Schalter `\Conc` nicht verwendet wird. Der Parameter thickness_2 enthält keinen gültigen Wert, da der Schalter `\Conc` nicht in der Instruktion `STClose` verwendet wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Rückgabewert

Datentyp: `bool`

TRUE, wenn das geprüfte Werkzeug geschlossen ist, d. h. die gewünschte Spitzenkraft erreicht wird.

FALSE, wenn das geprüfte Werkzeug nicht geschlossen ist.

Argumente

```
STIsClosed ToolName [\RetThickness]
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

`[\RetThickness]`

Datentyp: `num`

Erzielte Dicke [mm].

HINWEIS! Nur gültig, wenn `\Conc` in einer vorangehenden `STClose`-Instruktion verwendet wurde.

Syntax

```
STIsClosed '('
  [ ToolName ':=' ] < expression (IN) of string > ')'
  ['\' RetThickness ':=' < variable or persistent (INOUT) of num
       > ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Öffnen eines Servowerkzeugs	*STOpen - Öffnen eines Servowerkzeugs auf Seite 836*
Schließen eines Servowerkzeugs	*STClose - Schließen eines Servowerkzeugs auf Seite 817*
Prüfen, ob ein Servowerkzeug geöffnet ist	*STIsOpen - Prüfen, ob ein Servowerkzeug geöffnet ist auf Seite 1513*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.179 STIsIndGun - Überprüfen, ob sich ein Servowerkzeug im unabhängigen Modus befindet

Verwendung

Mit `STIsIndGun` wird überprüft, ob sich ein Servowerkzeug im unabhängigen Modus befindet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `STIsIndGun`.

Beispiel 1:

```
IF STIsIndGun(gun1) THEN
  ! Start the gun calibration
  STCalib gun1\TipChg;
ELSE
  ...
ENDIF
```

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn sich das überprüfte Werkzeug im unabhängigen Modus befindet, d. h. wenn die Zange unabhängig von den Bewegungen des Roboters bewegt werden kann.

`FALSE`, wenn sich das überprüfte Werkzeug *nicht* im unabhängigen Modus befindet.

Argumente

```
STIsIndGun ToolName
```

`ToolName`

Datentyp: `string`

Name der mechanischen Einheit

Syntax

```
STIsIndGun '('
  [ ToolName ':=' ] < expression (IN) of string > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Kalibrieren eines Servowerkzeugs	*STCalib - Kalibrieren eines Servowerkzeugs auf Seite 812*
Setzen der Zange in den unabhängigen Modus	*STIndGun - Versetzen des Servowerkzeug in den Modus „Unabhängig“. auf Seite 823*
Zurücksetzen der Zange aus dem Modus „Unabhängig“	*STIndGun - Zurücksetzen des Werkzeugs aus dem Modus „Unabhängig“. auf Seite 825*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.180 STIsOpen - Prüfen, ob ein Servowerkzeug geöffnet ist

Verwendung

`STIsOpen` wird verwendet, um zu prüfen, ob ein Servowerkzeug geöffnet ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `STIsOpen`.

Beispiel 1:

```
IF STIsOpen(gun1) THEN
  !Start the motion
  MoveL ...
ELSE
  ...
ENDIF
```

Es wird geprüft, ob die Zange offen ist.

Beispiel 2:

```
STCalib "sgun" \TipWear \Conc;
WHILE NOT(STIsOpen("sgun") \RetTipWear:=tipwear \RetPosAdj:=posadj)
     DO;
WaitTime 0.1;

ENDWHILE
```

IF tipwear > 20...

IF posadj > 25...

Eine Kalibrierung des Elektrodenverschleißes wird durchgeführt. Es wird darauf gewartet, dass die Zange `sgun` geöffnet ist. Der Elektrodenverschleiß und die Positionskorrekturwerte werden abgelesen.

Beispiel 3

Beispiele von unzulässigen Kombinationen:

```
STCalib "sgun" \TipWear \RetTipWear:=tipwear_1 \Conc;
WHILE NOT(STIsOpen("sgun") \RetTipWear:=tipwear_2) DO;
  WaitTime 0.1;

ENDWHILE
```

Eine Kalibrierung des Elektrodenverschleißes wird gestartet. Der Parameter `tipwear_1` enthält **keinen** gültigen Wert, weil der Schalter `\Conc` verwendet wird. Wenn die Kalibrierung fertig ist und `STIsOpen` den Wert `TRUE` zurückgibt, enthält der Parameter `tipwear_2` einen gültigen Wert.

```
STCalib "sgun" \TipWear \RetTipWear:=tipwear_1;

WHILE NOT(STIsOpen("sgun") \RetTipWear:=tipwear_2) DO;
  WaitTime 0.1;

ENDWHILE
```

Eine Kalibrierung des Elektrodenverschleißes wird durchgeführt. Der Parameter `tipwear_1`enthält einen gültigen Wert, weil der Schalter `\Conc` nicht verwendet

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.180 STIsOpen - Prüfen, ob ein Servowerkzeug geöffnet ist

Servo Tool Control

Fortsetzung

wird. Wenn STIsOpen den Wert TRUE zurückgibt, enthält der Parameter tipwear_2 keinen gültigen Wert, weil der Schalter \Conc nicht in STCalib verwendet wurde.

Rückgabewert

Datentyp: bool

TRUE, wenn das geprüfte Werkzeug offen ist, d. h., der Werkzeugarm befindet sich in der programmierten offenen Position.

FALSE, wenn das geprüfte Werkzeug nicht geöffnet ist.

Argumente

```
STIsOpen ToolName [\RetTipWear] [\RetPosAdj]
```

ToolName

Datentyp: string

Name der mechanischen Einheit

[\RetTipWear]

Datentyp: num

Erreichter Elektrodenverschleiß [mm]

Hinweis: Nur gültig, wenn \Conc in einer vorherigen STCalib-Instruktion verwendet wurde und wenn STIsOpen den Wert TRUE zurückgibt.

[\RetPosAdj]

Datentyp: num

Positionskorrektur seit letzter Kalibrierung [mm]

Hinweis: Nur gültig, wenn \Conc in einer vorherigen STCalib-Instruktion verwendet wurde und wenn STIsOpen den Wert TRUE zurückgibt.

Syntax

```
STIsOpen '('
  [ ToolName ':=' ] < expression (IN) of string > ')'
  [ '\' RetTipWear ':=' < variable or persistent(INOUT) of num >
       ] ';'
  [ '\' RetPosAdj ':=' < variable or persistent(INOUT) of num > ]
```

Eine Funktion mit einem Rückgabewert des Typs bool.

Weitere Informationen

Information über	Siehe
Öffnen eines Servowerkzeugs	*STOpen - Öffnen eines Servowerkzeugs auf Seite 836*
Schließen eines Servowerkzeugs	*STClose - Schließen eines Servowerkzeugs auf Seite 817*
Prüfen, ob ein Servowerkzeug geschlossen ist	*STIsClosed - Prüfen, ob ein Servowerkzeug geschlossen ist auf Seite 1510*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.181 StrDigCalc - Arithmetische Operationen mit dem Datentyp stringdig

Verwendung

Mit `StrDigCalc` werden arithmetische Operationen (+, -, *, /, %) an zwei Zeichenfolgen mit positiven Ziffern auf die gleiche Weise wie bei numerischen arithmetischen Operationen an positiven Ganzzahlen durchgeführt.

Mit dieser Funktion ist es möglich, positive Ganzzahlen (32-Bit Integer) oberhalb von 8.388.608 ("num") zu verarbeiten

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrDigCalc`.

Siehe auch *Weitere Beispiele auf Seite 1516*.

Beispiel 1:

```
res := StrDigCalc(str1, OpAdd, str2);
```

`res` wird dem Ergebnis der Additionsoperation für die Werte zugewiesen, die von den digitalen Zeichenfolgen `str1` und `str2` dargestellt werden.

Rückgabewert

Datentyp: `stringdig`

`stringdig` dient zum Darstellen großer positiver Ganzzahlen in einer Zeichenfolge, die nur Ziffern enthält.

Dieser Datentyp wurde eingeführt, weil der Datentyp `num` positive Ganzzahlen größer als 8.388.608 nicht genau darstellen kann.

Argumente

```
StrDigCalc (StrDig1 Operation StrDig2)
```

`StrDig1`

String Digit 1

Datentyp: `stringdig`

Zeichenfolge, die eine positive ganze Zahl darstellt.

`Operation`

Arithmetic operator

Datentyp: `opcalc`

Definiert die arithmetische Operation, die an zwei digitalen Zeichenfolgen durchgeführt werden soll. Die folgenden arithmetischen Operationen vom Datentyp `opcalc` können verwendet werden: `OpAdd`, `OpSub`, `OpMult`, `OpDiv` und `OpMod`.

`StrDig2`

String Digit 2

Datentyp: `stringdig`

Zeichenfolge, die eine positive ganze Zahl darstellt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.181 StrDigCalc - Arithmetische Operationen mit dem Datentyp stringdig
RobotWare - OS
Fortsetzung

Programmabarbeitung

Diese Funktion führt Folgendes aus:

- Überprüfen, ob in `StrDig1` und `StrDig2` nur die Ziffern 0 bis 9 verwendet werden
- Die beiden Ziffernzeichenfolgen in `long integers` konvertieren
- Arithmetische Operation an zwei `long integers` ausführen
- Das Ergebnis von `long integer` in `stringdig` konvertieren

Weitere Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `StrDigCalc`.

Beispiel 1:

```
res := StrDigCalc(str1, OpSub, str2);
```

`res` wird dem Ergebnis der Subtraktionsoperation für die Werte zugewiesen, die von den digitalen Zeichenfolgen `str1` und `str2` dargestellt werden.

Beispiel 2:

```
res := StrDigCalc(str1, OpMult, str2);
```

`res` wird dem Ergebnis der Multiplikationsoperation für die Werte zugewiesen, die von den digitalen Zeichenfolgen `str1` und `str2` dargestellt werden.

Beispiel 3

```
res := StrDigCalc(str1, OpDiv, str2);
```

`res` wird dem Ergebnis der Divisionsoperation für die Werte zugewiesen, die von den digitalen Zeichenfolgen `str1` und `str2` dargestellt werden.

Beispiel 4:

```
res := StrDigCalc(str1, OpMod, str2);
```

`res` wird dem Ergebnis der Modulo-Operation für die Werte zugewiesen, die von den digitalen Zeichenfolgen `str1` und `str2` dargestellt werden.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_INT_NOTVAL`	Eingangswerte, nicht nur Ziffern oder Modulo durch Null
`ERR_INT_MAXVAL`	Einganswert über 4294967295
`ERR_CALC_OVERFLOW`	Ergebnis außerhalb des Bereichs 0...4294967295
`ERR_CALC_NEG`	Negative Subtraktion, d. h. StrDig2 > StrDig1
`ERR_CALC_DIVZERO`	Division durch null

Einschränkungen

`StrDigCalc` akzeptiert nur Ziffernzeichenfolgen (Zeichen 0 bis 9). Alle anderen Zeichen in `stringdig` führen zu einem Fehler.

Diese Funktion kann nur positive Ganzzahl bis 4.294.967.295 behandeln.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.181 StrDigCalc - Arithmetische Operationen mit dem Datentyp stringdig

RobotWare - OS

Fortsetzung

Syntax

```
StrDigCalc '('
  [ StrDig1 ':=' ] < expression (IN) of stringdig > ','
  [ Operation ':=' ] < expression (IN) of opcalc > ','
  [ StrDig2 ':=' ] < expression (IN) of stringdig > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `stringdig`.

Weitere Informationen

Information über	Siehe
Zeichenfolge, die nur Ziffern enthält	*stringdig - Zeichenfolge, die nur Ziffern enthält auf Seite 1808*
Arithmetische Operatoren	*opcalc - Arithmetischer Operator auf Seite 1742*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.182 StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten

Verwendung

Mit `StrDigCmp` werden zwei Zeichenfolgen mit positiven Ziffern auf die gleiche Weise wie bei einem numerischen Vergleich von zwei positiven Ganzzahlen verglichen.

Mit dieser Funktion ist es möglich, positive Ganzzahlen (32-Bit Integer) oberhalb von 8.388.608 ("num") zu verarbeiten

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `StrDigCmp`.

Beispiel 1:

```
VAR stringdig digits1 := "1234";
VAR stringdig digits2 := "1256";
VAR bool is_equal;
is_equal := StrDigCmp(digits1, EQ, digits2);
```

Die Variable `is_equal` wird auf `FALSE` gesetzt, weil der numerische Wert `1234` ungleich `1256` ist.

Rückgabewert

Datentyp: `bool`

TRUE, wenn die angegebene Bedingung zutrifft, andernfalls FALSE.

Argumente

```
StrDigCmp (StrDig1 Relation StrDig2)
```

`StrDig1`

String Digit 1

Datentyp: `stringdig`

Die erste numerisch zu vergleichende Zeichenfolge, die nur Ziffern enthält.

`Relation`

Datentyp: `opnum`

Definiert, wie die beiden Ziffernzeichenfolgen verglichen werden sollen. Die folgenden vordefinierten Konstanten vom Datentyp `opnum` können verwendet werden: `LT`, `LTEQ`, `EQ`, `NOTEQ`, `GTEQ` oder `GT`.

`StrDig2`

String Digit 2

Datentyp: `stringdig`

Die zweite numerisch zu vergleichende Zeichenfolge, die nur Ziffern enthält.

Programmabarbeitung

Diese Funktion führt Folgendes aus:

- Überprüfen, ob in `StrDig1` und `StrDig2` nur die Ziffern 0 bis 9 verwendet werden

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Die beiden Ziffernzeichenfolgen in `long integers` konvertieren
- Die beiden `long integers` numerisch vergleichen

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_INT_NOTVAL`	Eingangswerte, nicht nur Ziffern
`ERR_INT_MAXVAL`	Wert über 4294967295

Einschränkungen

`StrDigCmp` akzeptiert nur Ziffernzeichenfolgen (Zeichen 0 bis 9). Alle anderen Zeichen in `stringdig` führen zu einem Fehler.

Diese Funktion kann nur positive Ganzzahl bis 4.294.967.295 behandeln.

Syntax

```
StrDigCmp '('
  [ StrDig1 ':=' ] < expression (IN) of stringdig > ','
  [ Relation ':=' ] < expression (IN) of opnum > ','
  [ StrDig2 ':=' ] < expression (IN) of stringdig > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Zeichenfolge, die nur Ziffern enthält	*stringdig - Zeichenfolge, die nur Ziffern enthält auf Seite 1808*
Vergleichsoperatoren	*opnum - Vergleichsoperator auf Seite 1743*
Dateizeitinformationen	*FileTimeDnum - Abrufen von Zeitinformationen über eine Datei auf Seite 1307*
Zeitpunkt der Änderung des geladenen Moduls	*ModTimeDnum - Abrufen des Dateiänderungszeitpunkts für das geladene Modul auf Seite 1400*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.183 StrFind - Suchen nach einem Zeichen in einer Zeichenfolge

Verwendung

`StrFind` (*String Find*) wird verwendet, um in einer Zeichenfolge beginnend bei einer angegebenen Position nach einem Zeichen zu suchen, das einer angegebenen Zeichenmenge angehört.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrFind`.

Beispiel 1:

```
VAR num found;
found := StrFind("Robotics",1,"aeiou");
```

Die Variable `found` erhält den Wert 2.

```
found := StrFind("Robotics",1,"aeiou"\NotInSet);
```

Die Variable `found` erhält den Wert 1.

```
found := StrFind("IRB 6700",1,STR_DIGIT);
```

Die Variable `found` erhält den Wert 5.

```
found := StrFind("IRB 6700",1,STR_WHITE);
```

Die Variable `found` erhält den Wert 4.

Rückgabewert

Datentyp: `num`

Die Zeichenposition des ersten Zeichens an oder nach der angegebenen Position, das der angegebenen Zeichenmenge angehört. Wenn kein entsprechendes Zeichen gefunden wird, wird die Zeichenfolgenlänge +1 zurückgegeben.

Argumente

```
StrFind (Str ChPos Set [\NotInSet])
```

`Str`

String

Datentyp: `string`

Die Zeichenfolge, in der gesucht wird.

`ChPos`

Character Position

Datentyp: `num`

Position des Anfangszeichens. Ein Laufzeitfehler tritt auf, wenn die Position außerhalb der Zeichenfolge liegt.

`Set`

Datentyp: `string`

Zeichenmenge, die zum Vergleich dient. Siehe auch *Vordefinierte Daten auf Seite 1521*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\NotInSet]`

Datentyp: `switch`

Suche nach einem Zeichen, das nicht in `Set` enthalten ist.

Syntax

```
StrFind '('
  [Str ':='] <expression (IN) of string>','
  [ChPos ':='] <expression (IN) of num>','
  [Set ':='] <expression (IN) of string>
  ['\' NotInSet]')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Vordefinierte Daten

Eine Reihe vordefinierter string-Konstanten ist im System verfügbar und kann mit string-Funktionen verwendet werden.

Name	Zeichensatz
`STR_DIGIT`	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
`STR_UPPER`	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
`STR_LOWER`	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
`STR_WHITE`	<blank character> ::=

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.184 StrFormat - Formatzeichenfolge.

Verwendung

`StrFormat` wird für die Formatierung einer Zeichenfolge verwendet.

Grundlegende Beispiele

Folgendes Beispiel veranschaulich die Funktion `StrFormat`:

Beispiel 1:

```
VAR string text1 := "This is a {1} and this is {2}";
...
TPWrite StrFormat(text1 \Arg1:="robot" \Arg2:="fast");
```

Die in den optionalen Argumenten `Arg1` und `Arg2` verwendeten Zeichenfolgen ersetzen `{1}` und `{2}`. Die Ausgabe ist dann:

This is a robot and this is fast

Rückgabewert

Datentyp: `string`

Argumente

```
StrFormat ( Text [\Arg1] [\Arg2] [\Arg3] [\Arg4] [\Arg5] [\Arg6]
     )
```

`Text`

Datentyp: `string`

Die zu formatierende Zeichenfolge.

`[\Arg1]`

Datentyp: `string`

Wenn die Zeichenfolge `{1}`, enthält, dann wird `{1}` durch die Zeichenfolge in diesem Argument ersetzt.

`[\Arg2]`

Datentyp: `string`

Wenn die Zeichenfolge `{2}`, enthält, dann wird `{2}` durch die Zeichenfolge in diesem Argument ersetzt.

`[\Arg3]`

Datentyp: `string`

Wenn die Zeichenfolge `{3}`, enthält, dann wird `{3}` durch die Zeichenfolge in diesem Argument ersetzt.

`[\Arg4]`

Datentyp: `string`

Wenn die Zeichenfolge `{4}`, enthält, dann wird `{4}` durch die Zeichenfolge in diesem Argument ersetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`[\Arg5]`

Datentyp: `string`

Wenn die Zeichenfolge `{5}`, enthält, dann wird `{5}` durch die Zeichenfolge in diesem Argument ersetzt.

`[\Arg6]`

Datentyp: `string`

Wenn die Zeichenfolge `{6}`, enthält, dann wird `{6}` durch die Zeichenfolge in diesem Argument ersetzt.

Einschränkungen

Die gesamte Stringlänge in RAPID ist 80. Dies gilt für alle Eingaben als Ergebnis der Funktion.

Syntax

```
StrFormat
  [Text ':='] <expression (IN) of string>
  ['\' Arg1 ':=' <expression (IN) of string>]
  ['\' Arg2 ':=' <expression (IN) of string>]
  ['\' Arg3 ':=' <expression (IN) of string>]
  ['\' Arg4 ':=' <expression (IN) of string>]
  ['\' Arg5 ':=' <expression (IN) of string>]
  ['\' Arg6 ':=' <expression (IN) of string>]')'
```

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Datentyp `string`	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.185 StrLen - Abrufen der Zeichenfolgenlänge

Verwendung

`StrLen` (*String Length*) wird verwendet, um die aktuelle Länge einer Zeichenfolge zu ermitteln.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrLen`.

Beispiel 1:

```
VAR num len;
len := StrLen("Robotics");
```

Die Variable `len` erhält den Wert `8`.

Rückgabewert

Datentyp: `num`

Die Anzahl der Zeichen in der Zeichenfolge (>=0).

Argumente

```
StrLen (Str)
```

`Str`

String

Datentyp: `string`

Die Zeichenfolge, in der die Anzahl der Zeichen gezählt werden soll.

Syntax

```
StrLen '('
  [Str ':='] <expression (IN) of string>')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Instruktionen, Funktionen und Datentypen*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Instruktionen, Funktionen und Datentypen*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.186 StrMap - Zuordnen einer Zeichenfolge

Verwendung

`StrMap` (*String Mapping*) wird verwendet, um eine Kopie einer Zeichenfolge zu erstellen, in der alle Zeichen gemäß einer bestimmten Zuordnung umgesetzt werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `StrMap`.

Beispiel 1:

```
VAR string str;
str := StrMap("Robotics","aeiou","AEIOU");
```

Die Variable `str` erhält den Wert `RObOtIcs`.

Beispiel 2:

```
str := StrMap("Robotics",STR_LOWER, STR_UPPER);
```

Die Variable `str` erhält den Wert `ROBOTICS`.

Rückgabewert

Datentyp: `string`

Die Zeichenfolge, die durch Zuordnen der Zeichen in der angegebenen Zeichenfolge entsteht, wie durch die From- und To-Zeichenfolge angegeben. Jedes Zeichen aus der angegebenen Zeichenfolge, das sich in der From-Zeichenfolge befindet, wird durch ein Zeichen an der entsprechenden Position in der To-Zeichenfolge ersetzt. Zeichen, für die keine Zuordnung definiert ist, werden unverändert in die Ergebniszeichenfolge kopiert.

Argumente

```
StrMap ( Str FromMap ToMap)
```

`Str`

String

Datentyp: `string`

Die zuzuordnende Zeichenfolge.

`FromMap`

Datentyp: `string`

Indexteil der Zuordnung. Siehe auch *Vordefinierte Daten auf Seite 1526*.

`ToMap`

Datentyp: `string`

Wertteil der Zuordnung. Siehe auch *Vordefinierte Daten auf Seite 1526*.

Syntax

```
StrMap '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ FromMap ':=' ] <expression (IN) of string> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.186 StrMap - Zuordnen einer Zeichenfolge

RobotWare - OS

Fortsetzung

```
[ ToMap ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Vordefinierte Daten

Eine Reihe vordefinierter string-Konstanten ist im System verfügbar und kann mit string-Funktionen verwendet werden.

Name	Zeichensatz
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.187 StrMatch - Suchen nach einem Muster in einer Zeichenfolge

Verwendung

`StrMatch` (*String Match*) wird verwendet, um in einer Zeichenfolge beginnend bei einer angegebenen Position nach einem angegebenen Muster zu suchen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrMatch`.

Beispiel 1:

```
VAR num found;

found := StrMatch("Robotics",1,"bo");
```

Die Variable `found` erhält den Wert `3`.

Rückgabewert

Datentyp: `num`

Die Zeichenposition der ersten Teilzeichenfolge an oder nach der angegebenen Position, die mit der angegebenen Muster-Zeichenfolge identisch ist. Wenn keine entsprechende Teilzeichenfolge gefunden wird, wird die Zeichenfolgenlänge +1 zurückgegeben.

Argumente

```
StrMatch (Str ChPos Pattern)
```

`Str`

String

Datentyp: `string`

Die Zeichenfolge, in der gesucht wird.

`ChPos`

Character Position

Datentyp: `num`

Position des Anfangszeichens. Ein Laufzeitfehler tritt auf, wenn die Position außerhalb der Zeichenfolge liegt.

`Pattern`

Datentyp: `string`

Die Musterzeichenfolge, die gesucht wird.

Syntax

```
StrMatch '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ ChPos ':=' ] <expression (IN) of num> ','
  [ Pattern ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.187 StrMatch - Suchen nach einem Muster in einer Zeichenfolge

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.188 StrMemb - Prüfen, ob ein Zeichen einer Zeichenmenge angehört

Verwendung

`StrMemb` (*String Member*) wird verwendet, um zu prüfen, ob ein angegebenes Zeichen in einer Zeichenfolge einer bestimmten Zeichenmenge angehört.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrMemb`.

Beispiel 1:

```
VAR bool memb;
memb := StrMemb("Robotics",2,"aeiou");
```

Die Variable `memb` erhält den Wert `TRUE`, weil `o` in der Zeichenmenge `"aeiou"` vorhanden ist.

```
memb := StrMemb("Robotics",3,"aeiou");
```

Die Variable `memb` erhält den Wert `FALSE`, weil `b` in der Zeichenmenge `"aeiou"` nicht vorhanden ist.

```
memb := StrMemb("S-721 68 VÄSTERÅS",3,STR_DIGIT);
```

Die Variable `memb` erhält den Wert `TRUE`, weil `7` in der Zeichenmenge `STR_DIGIT` vorhanden ist.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn das Zeichen an der angegebenen Position der angegebenen Zeichenmenge angehört.

Argumente

```
StrMemb (Str ChPos Set)
```

`Str`

String

Datentyp: `string`

Die Zeichenfolge, in der abgefragt wird.

`ChPos`

Character Position

Datentyp: `num`

Die Zeichenposition, die abgefragt wird. Ein Laufzeitfehler tritt auf, wenn die Position außerhalb der Zeichenfolge liegt.

`Set`

Datentyp: `string`

Zeichenmenge, die zum Vergleich dient.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.188 StrMemb - Prüfen, ob ein Zeichen einer Zeichenmenge angehört

RobotWare - OS

Fortsetzung

Vordefinierte Daten

Eine Reihe vordefinierter string-Konstanten ist im System verfügbar und kann mit string-Funktionen verwendet werden.

Name	Zeichensatz
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Syntax

```
StrMemb '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ ChPos ':=' ] <expression (IN) of num> ','
  [ Set ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.189 StrOrder - Prüfen, ob Zeichenfolgen geordnet sind

Verwendung

`StrOrder` (*String Order*) vergleicht zwei Zeichenfolgen (Zeichen für Zeichen) und gibt einen booleschen Wert zurück. Dieser gibt an, ob die Zeichen in der Zeichenfolge in einer angegebenen Reihenfolge angeordnet sind.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `StrOrder`.

Beispiel 1:

```
VAR bool le;

le := StrOrder("FIRST","SECOND",STR_UPPER);
```

Die Variable `le` erhält den Wert `TRUE`, weil "`F`" in der Zeichenreihenfolge `STR_UPPER` vor "`S`" steht.

Beispiel 2:

```
VAR bool le;

le := StrOrder("FIRST","FIRSTB",STR_UPPER);
```

Die Variable `le` erhält den Wert `TRUE`, weil `"FIRSTB"` ein zusätzliches Zeichen in der Zeichenreihenfolge enthält („kein Zeichen“ verglichen mit `"B"`).

Beispiel 3

```
VAR bool le;

le := StrOrder("FIRSTB","FIRST",STR_UPPER);
```

Die Variable `le` erhält den Wert `FALSE`, weil `"FIRSTB"` ein zusätzliches Zeichen in der Zeichenreihenfolge enthält (`"B"` verglichen mit keinem Zeichen).

Rückgabewert

Datentyp: `bool`

`TRUE`, falls die erste vor der zweiten Zeichenfolge (`Str1` <= `Str2`) kommt, wenn die Zeichen wie angegeben geordnet sind.

Zeichen, die nicht in der definierten Ordnung enthalten sind, folgen den darin enthaltenen Zeichen.

Argumente

```
StrOrder ( Str1 Str2 Order)
```

`Str1`

String 1

Datentyp: `string`

Erster Zeichenfolgenwert.

`Str2`

String 2

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.189 StrOrder - Prüfen, ob Zeichenfolgen geordnet sind
RobotWare - OS
Fortsetzung

Datentyp: `string`

Zweiter Zeichenfolgenwert.

`Order`

Datentyp: `string`

Folge der Zeichen, die die Ordnung definieren. Siehe auch *Vordefinierte Daten auf Seite 1532*.

Vordefinierte Daten

Eine Reihe vordefinierter string-Konstanten ist im System verfügbar und kann mit string-Funktionen verwendet werden.

Name	Zeichensatz
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

Syntax

```
StrOrder '('
  [ Str1 ':=' ] <expression (IN) of string> ','
  [ Str2 ':=' ] <expression (IN) of string> ','
  [ Order ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.190 StrPart - Suchen eines Teils der Zeichenfolge

Verwendung

`StrPart` (*String Part)*) wird verwendet, um einen Teil einer Zeichenfolge als neue Zeichenfolge zu finden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrPart`.

Beispiel 1:

```
VAR string part;
part := StrPart("Robotics",1,5);
```

Die Variable `part` erhält den Wert `"Robot"`.

Rückgabewert

Datentyp: `string`

Die Teilzeichenfolge der angegebenen Zeichenfolge mit der angegebenen Länge und dem Beginn an der angegebenen Zeichenposition.

Argumente

```
StrPart (Str ChPos Len)
```

`Str`

String

Datentyp: `string`

Die Zeichenfolge, in der ein Teil gefunden werden soll.

`ChPos`

Character Position

Position des Anfangszeichens. Ein Laufzeitfehler tritt auf, wenn die Position außerhalb der Zeichenfolge liegt.

`Len`

Länge

Datentyp: `num`

Länge der Teilzeichenfolge. Ein Laufzeitfehler tritt auf, wenn die Länge negativ ist oder die Länge der Zeichenfolge überschreitet oder wenn die Teilzeichenfolge (teilweise) außerhalb der Zeichenfolge liegt.

Syntax

```
StrPart '('
[ Str ':=' ] <expression (IN) of string> ','
[ ChPos ':=' ] <expression (IN) of num> ','
[ Len ':=' ] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.190 StrPart - Suchen eines Teils der Zeichenfolge

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.191 StrToByte - Konvertieren von string-Daten in Byte-Daten

Verwendung

`StrToByte` (*String To Byte*) wird verwendet, um eine Zeichenfolge mit einem definierten Byte-Datenformat in den Datentyp "byte" zu konvertieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrToByte`.

Beispiel 1:

```
VAR string con_data_buffer{5} := ["10", "AE", "176", "00001010",
     "A"];
VAR byte data_buffer{5};
data_buffer{1} := StrToByte(con_data_buffer{1});
```

Der Inhalt der Datenfeldkomponente `data_buffer{1}` ist nach Abarbeiten der Funktion `StrToByte ...` 10 dezimal.

```
data_buffer{2} := StrToByte(con_data_buffer{2}\Hex);
```

Der Inhalt der Datenfeldkomponente `data_buffer{2}` ist nach Abarbeiten der Funktion `StrToByte ...` 174 dezimal.

```
data_buffer{3} := StrToByte(con_data_buffer{3}\Okt);
```

Der Inhalt der Datenfeldkomponente `data_buffer{3}` ist nach Abarbeiten der Funktion `StrToByte ...` 126 dezimal.

```
data_buffer{4} := StrToByte(con_data_buffer{4}\Bin);
```

Der Inhalt der Datenfeldkomponente `data_buffer{4}` ist nach Abarbeiten der Funktion `StrToByte ...` 10 dezimal.

```
data_buffer{5} := StrToByte(con_data_buffer{5}\Char);
```

Der Inhalt der Datenfeldkomponente `data_buffer{5}` ist nach Abarbeiten der Funktion `StrToByte ...` 65 dezimal.

Rückgabewert

Datentyp: `byte`

Das Ergebnis der Konvertierung in Dezimaldarstellung.

Argumente

```
StrToByte (ConStr [\Hex] | [\Okt] | [\Bin] | [\Char])
```

`ConStr`

Convert String

Datentyp: `string`

Die zu konvertierenden string-Daten.

Wenn das optionale switch-Argument weggelassen wird, haben die zu konvertierenden string-Daten das `decimal`-Format (Dec).

`[\Hex]`

Hexadecimal

Datentyp: `switch`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.191 StrToByte - Konvertieren von string-Daten in Byte-Daten

RobotWare - OS

Fortsetzung

Der zu konvertierende String hat `hexadecimal`-Format.

`[\Okt]`

Octal

Datentyp: `switch`

Der zu konvertierende String hat `octal`-Format.

`[\Bin]`

Binary

Datentyp: `switch`

Der zu konvertierende String hat `binary`-Format.

`[\Char]`

Character

Datentyp: `switch`

Der zu konvertierende String hat `ASCII`-Format.

Einschränkungen

Abhängig vom Format der zu konvertierenden Zeichenfolge sind folgende String-Daten gültig:

Format	Zeichenfolgenlänge:	Bereich
Dec .....: '0'- '9'	3	"0" - "255"
Hex .....: '0'- '9', 'a'-'f', 'A'- 'F'	2	"0" - "FF"
Oct .....: '0'- '7'	3	"0" - "377"
Bin .....: '0'- '1'	8	"0" - "11111111"
Char ....: Beliebiges ASCII-Zeichen	1	Ein ASCII-Zeichen

RAPID-Zeichencodes (z. B. „\07“ für BEL-Steuerzeichen) können als Argumente in `ConStr` verwendet werden.

Syntax

```
StrToByte '('
  [ConStr ':=' ] <expression (IN) of string>
  ['\' Hex ] | ['\' Okt] | ['\' Bin] | ['\' Char]
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `byte`.

Weitere Informationen

Information über	Siehe
Konvertieren von byte-Daten in string-Daten	*ByteToStr - Konvertieren von byte-Daten in string-Daten auf Seite 1218*
Andere Bit- (Byte-) Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Andere String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.192 StrToVal - Konvertieren einer Zeichenfolge in einen Wert

Verwendung

`StrToVal` (*String To Value*) wird verwendet, um eine Zeichenfolge in einen Wert eines beliebigen Datentyps zu konvertieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `StrToVal`.

Siehe auch *Weitere Beispiele auf Seite 1537*.

Beispiel 1:

```
VAR bool ok;
VAR num nval;
ok := StrToVal("3.85",nval);
```

Die Variable `ok` erhält den Wert `TRUE` und `nval` erhält den Wert `3.85`.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn die angeforderte Konvertierung erfolgreich war, sonst `FALSE`.

Argumente

```
StrToVal ( Str Val )
```

`Str`

String

Datentyp: `string`

Ein Zeichenfolgenwert mit literalen Daten, deren Format dem in Argument `Val` verwendeten Datentyp entspricht. Gültiges Format wie für Literal-Aggregate in RAPID.

`Val`

Value

Datentyp: `anytype`

Name der Variablen oder Speichernden eines beliebigen Datentyps zur Speicherung des Konvertierungsergebnisses.

Es kann ein beliebiger Typ von value-Daten verwendet werden, z. B. Einzeldaten, Datensätze, Datensatzkomponenten, Datenfelder oder Datenfeldkomponenten. Die Daten bleiben unverändert, wenn die angeforderte Konvertierung fehlgeschlagen ist, weil das Format nicht mit den Daten im Argument `Str` übereinstimmt.

Weitere Beispiele

Weitere Beispiele für die Funktion `StrToVal` sind unten aufgeführt.

Beispiel 1:

```
VAR string str15 := "[600, 500, 225.3]";
VAR bool ok;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.192 StrToVal - Konvertieren einer Zeichenfolge in einen Wert

RobotWare - OS

Fortsetzung

```
VAR pos pos15;

ok := StrToVal(str15,pos15);
```

Die Variable `ok` erhält den Wert `TRUE` und die Variable `pos15` erhält den Wert, der in der Zeichenfolge `str15` angegeben ist.

Syntax

```
StrToVal '('
  [ Str ':=' ] <expression (IN) of string> ','
  [ Val ':=' ] <var or pers (INOUT) of anytype>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.193 Tan - Berechnet den Tangens.

Verwendung

`Tan` (*Tangent*) wird zur Berechnung des Tangenswerts aus einem Winkel verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `Tan`.

Beispiel 1:

```
VAR num angle;
VAR num value;
...
...
value := Tan(angle);
```

`value` ruft den Tangenswert von `angle` ab.

Rückgabewert

Datentyp: `num`

Der Tangenswert.

Argumente

```
Tan (Angle)
```

`Angle`

Datentyp: `num`

Der Wert des Winkels in Grad.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_TAN_DEGREES_NOTVAL`	Winkel 90 und 270 Grad ist undefiniert für tan().

Syntax

```
Tan '('
  [Angle ':='] <expression (IN) of num>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Arcustangens mit Rückgabewert im Bereich [-180, 180]	*ATan2 - Berechnen des Arcustangens2 auf Seite 1187*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.194 TanDnum - Berechnen des Tangens

Verwendung

`TanDnum` (*Tangent*) wird zur Berechnung des Tangenswertes aus einem Winkel verwendet `dnum`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TanDnum`.

Beispiel 1:

```
VAR dnum angle;
VAR dnum value;
...
...
value := TanDnum(angle);
```

`value` ruft den Tangenswert von `angle` ab.

Rückgabewert

Datentyp: `dnum`

Der Tangenswert.

Argumente

```
TanDnum (Angle)
```

`Angle`

Datentyp: `dnum`

Der Wert des Winkels in Grad.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_TAN_DEGREES_NOTVAL`	Winkel 90 und 270 Grad ist undefiniert für tan().

Syntax

```
TanDnum '('
  [Angle ':='] <expression (IN) of dnum>
  ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Arcustangens mit Rückgabewert im Bereich [-180, 180]	*ATan2Dnum - Berechnen des Arcustangens2 auf Seite 1188*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.195 TaskRunMec - Überprüfen, ob die Task eine mechanische Einheit steuert

Verwendung

`TaskRunMec` wird verwendet, um zu prüfen, ob die Programmtask eine mechanische Einheit (Roboter mit TCP oder Manipulator ohne TCP) steuert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TaskRunMec`.

Beispiel 1:

```
VAR bool flag;
...
flag := TaskRunMec( );
```

Wenn die aktuelle Task eine mechanische Einheit steuert, hat `flag` den Wert `TRUE`, anderenfalls `FALSE`.

Rückgabewert

Datentyp: `bool`

Wenn die aktuelle Task eine mechanische Einheit steuert, ist der Rückgabewert `TRUE`, anderenfalls `FALSE`.

Programmabarbeitung

Prüfen, ob die Programmtask eine mechanische Einheit steuert.

Syntax

```
TaskRunMec '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Prüfen, ob die Task einen Roboter steuert	*TaskRunRob - Überprüfen, ob die Task einen Roboter steuert auf Seite 1542*
Aktivieren und Deaktivieren mechanischer Einheiten	*ActUnit - Aktivieren einer mechanischen Einheit auf Seite 30* *DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Konfiguration von mechanischen Einheiten	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.196 TaskRunRob - Überprüfen, ob die Task einen Roboter steuert

Verwendung

`TaskRunRob` wird verwendet, um zu prüfen, ob die Programmtask einen Roboter (mechanische Einheit mit TCP) steuert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TaskRunRob`.

Beispiel 1:

```
VAR bool flag;
...
flag := TaskRunRob();
```

Wenn die aktuelle Task einen Roboter steuert, wird `flag` auf `TRUE` gesetzt, anderenfalls auf `FALSE`.

Rückgabewert

Datentyp: `bool`

Wenn die aktuelle Task einen Roboter steuert, ist der Rückgabewert `TRUE`, anderenfalls `FALSE`.

Programmabarbeitung

Prüfen, ob die Programmtask einen Roboter steuert.

Syntax

```
TaskRunRob '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Prüfen, ob die Task eine mechanische Einheit steuert	*TaskRunMec - Überprüfen, ob die Task eine mechanische Einheit steuert auf Seite 1541*
Aktivieren und Deaktivieren mechanischer Einheiten	*ActUnit - Aktivieren einer mechanischen Einheit auf Seite 30* *DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Konfiguration von mechanischen Einheiten	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.197 TasksInSync – Gibt die Anzahl der synchronisierten Tasks zurück

Verwendung

`TasksInSync` wird verwendet, um die Anzahl der synchronisierten Tasks abzurufen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TaskInSync`.

Beispiel 1:

```
VAR tasks tasksInSyncList{6};
...

PROC main ()
  VAR num noOfSynchTasks;
  ...
  noOfSynchTasks:= TasksInSync (tasksInSyncList);
  TPWrite "No of synchronized tasks = "\Num:=noOfSynchTasks;
ENDPROC
```

Der Variablen `noOfSynchTasks` wird die Anzahl der synchronisierten Tasks zugewiesen und die `tasksInSyncList` enthält die Namens der synchronisierten Tasks. In diesem Beispiel ist die Taskliste eine Variable, aber sie kann auch eine Persistente sein.

Rückgabewert

Datentyp: `num`

Die Anzahl der synchronisierten Tasks.

Argumente

```
TaskInSync (TaskList)
```

`TaskList`

Datentyp: `tasks`

Inout-Argument, das in einer Taskliste (Datenfeld) den Namen (`string`) der synchronisierten Programmtasks darstellt. Die Taskliste kann entweder vom Typ `VAR` oder `PERS` sein.

Programmabarbeitung

Die Funktion gibt die Anzahl der synchronisierten Tasks im System zurück. Die Namen der synchronisierten Tasks werden im Inout-Argument TaskList dargestellt. Falls es keine synchronisierten Tasks gibt, enthält die Liste nur leere Zeichenfolgen.

Einschränkungen

Derzeit wird nur eine synchronisierte Gruppe unterstützt, weshalb `TasksInSync` die Anzahl der in dieser Gruppe synchronisierten Tasks zurückgibt.

Syntax

```
TasksInSync
  [ TaskList ':=' ] < var or pers array {*} (INOUT) of tasks> ','
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.197 TasksInSync – Gibt die Anzahl der synchronisierten Tasks zurück

RobotWare - OS

Fortsetzung

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.198 TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist

Verwendung

`TaskIsActive` wird verwendet, um zu überprüfen, ob die normale Programmtask im **Taskauswahlfeld** auf dem FlexPendant aktiviert oder deaktiviert ist.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `TaskIsActive`.

Beispiel 1:

```
IF TaskIsActive("T_ROB1") = TSP_STATUS_ACT THEN
  TPWrite "T_ROB1 is active in the Task Selection Panel";
ENDIF
```

Überprüfen Sie, ob die Programmtask `T_ROB1` im **Taskauswahlfeld** auf dem FlexPendant aktiv ist.

Rückgabewert

Datentyp: `tsp_status`

Der aktuelle Status des Taskauswahlfeldes.

Argumente

```
TaskIsActive (TaskRef | TaskName)
```

`TaskRef`

Datentyp: `taskid`

Die Programmtask-Identität der Task, die ausgewählt werden soll.

Die vordefinierten Variablen des Datentyps `taskid` sind für alle Programmtasks im System verfügbar.

Die Variablenidentität ist *"taskname"+"Id"*, z. B. ist die Variablenidentität für die `T_ROB1`-Task `T_ROB1Id`.

`TaskName`

Datentyp: `string`

Der Programmtaskname der Task, die ausgewählt werden soll.

Vordefinierte Daten

Folgende vordefinierte, symbolische Konstanten des Typs `tsp_status` können zur Überprüfung des Rückgabewerts verwendet werden:

```
CONST tsp_status TSP_STATUS_NOT_NORMAL_TASK := 0;
CONST tsp_status TSP_STATUS_DEACT := 1;
CONST tsp_status TSP_STATUS_DEACT_SERV_ROUT := 2;
CONST tsp_status TSP_STATUS_ACT := 3;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.198 TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist

RobotWare - OS

Fortsetzung

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

ERR_TASKNAME	Der Name der Programmtask im Argument \TaskName kann im System nicht gefunden werden.

Syntax

```
TaskIsActive '('
  [ TaskRef ':=' ] <variable (VAR) of taskid>
  |[ TaskName ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs tsp_status.

Weitere Informationen

Information über	Siehe
Status des Taskauswahlfeldes	*tsp_status – Status des Taskauswahlfeldes auf Seite 1846*
Rufen Sie den aktuellen Status des Taskauswahlfeldes ab	*GetTSPStatus – Rufen Sie den aktuellen Status des Taskauswahlfeldes ab auf Seite 1344*
Überprüfen Sie, ob die Task ausgeführt wird	*TaskIsExecuting – Überprüfen Sie, ob die Task ausgeführt wird auf Seite 1547*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.199 TaskIsExecuting – Überprüfen Sie, ob die Task ausgeführt wird

Verwendung

`TaskIsExecuting` wird verwendet, um zu überprüfen, ob eine Programmtask ausgeführt wird.

Grundlegende Beispiele

Folgende Beispiele veranschaulichen die Funktion `TaskIsExecuting`.

Beispiel 1:

```
TPWrite "T_ROB1 is executing: " \Bool:=TaskIsExecuting("T_ROB1");
TPWrite "T_ROB2 is executing: " \Bool:=TaskIsExecuting("T_ROB2");
```

Überprüft, ob die Programmtasks `T_ROB1` und `T_ROB2` ausgeführt werden und schreibt den Wert auf den FlexPendant.

Rückgabewert

Datentyp: `bool`

Wenn die Programmtask ausgeführt wird, ist der Rückgabewert `TRUE`, andernfalls `FALSE`.

Argumente

```
TaskIsExecuting (TaskRef | TaskName)
```

`TaskRef`

Datentyp: `taskid`

Die Programmtask-Identität der Task, die ausgewählt werden soll.

Die vordefinierten Variablen des Datentyps `taskid` sind für alle Programmtasks im System verfügbar.

Die Variablenidentität ist *"taskname"+"Id"*, z. B. ist die Variablenidentität für die `T_ROB1`-Task `T_ROB1Id`.

`TaskName`

Datentyp: `string`

Der Programmtaskname der Task, die ausgewählt werden soll.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

`ERR_TASKNAME`	Der Name der Programmtask im Argument `\TaskName` kann im System nicht gefunden werden.

Syntax

```
TaskIsExecuting '('
  [ TaskRef ':=' ] <variable (VAR) of taskid>
  |[ TaskName ':=' ] <expression (IN) of string> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.199 TaskIsExecuting – Überprüfen Sie, ob die Task ausgeführt wird

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Prüfen Sie, ob eine normale Task aktiv ist	*TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist auf Seite 1545*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.200 TestAndSet - Prüfen und ggf. Setzen einer nicht gesetzten Variablen

Verwendung

`TestAndSet` kann gemeinsam mit einem normalen Datenobjekt des Typs `bool` als binäres Semaphor verwendet werden, um das exklusive Zugriffsrecht auf bestimmte RAPID-Codebereiche oder Systemressourcen zu beziehen. Diese Funktion kann sowohl zwischen unterschiedlichen Tasks als auch unterschiedlichen Abarbeitungsebenen (Interrupt- oder Ereignisroutinen) innerhalb derselben Task verwendet werden.

Beispiele für Ressourcen, die vor gleichzeitigem Zugriff geschützt werden müssen:

- RAPID-Routinen mit Funktionsproblemen bei simultaner Abarbeitung
- Bedienerprotokoll am FlexPendant

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TestAndSet`.

Siehe auch *Weitere Beispiele auf Seite 1550*.

Beispiel 1:

`MAIN`-Programmtask:

```
PERS bool tproutine_inuse := FALSE;
...
WaitUntil TestAndSet(tproutine_inuse);
TPWrite "First line from MAIN";
TPWrite "Second line from MAIN";
TPWrite "Third line from MAIN";
tproutine_inuse := FALSE;
```

`BACK1`-Programmtask:

```
PERS bool tproutine_inuse := FALSE;
...
WaitUntil TestAndSet(tproutine_inuse);
TPWrite "First line from BACK1";
TPWrite" Second line from BACK1";
TPWrite "Third line from BACK1";
tproutine_inuse := FALSE;
```

Damit im Bedienerprotokoll die Zeilen von `MAIN` und `BACK1` nicht verwechselt werden, garantiert die Funktion `TestAndSet`, dass alle drei Zeilen von jeder Task zusammenhängend bleiben und nicht getrennt werden.

Wenn die Vordergrundtask `MAIN` das Semaphor `TestAndSet(tproutine_inuse)` zuerst nimmt, muss die Hintergrundtask `BACK1` warten, bis die Vordergrundtask `MAIN` das Semaphor verlässt.

Rückgabewert

Datentyp: `bool`

`TRUE`, wenn das Semaphor vom Ausführenden der Funktion `TestAndSet` genommen wurde, sonst `FALSE`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.200 TestAndSet - Prüfen und ggf. Setzen einer nicht gesetzten Variablen

RobotWare - OS

Fortsetzung

Argumente

```
TestAndSet Object
```

`Object`

Datentyp: `bool`

Anwenderdefiniertes Datenobjekt, das als Semaphor verwendet werden soll. Das Datenobjekt kann den Typ VAR oder PERS aufweisen. Wenn `TestAndSet` zwischen verschiedenen Programmtasks verwendet wird, muss das Objekt vom Typ PERS oder eine installierte VAR (Objekt zwischen Tasks) sein.

Programmabarbeitung

Die Funktion prüft die anwenderdefinierte Variable, setzt sie (falls nicht gesetzt) und gibt `TRUE` zurück. Oder sie gibt `FALSE` zurück (falls die Variable gesetzt ist). Alle diese Aktionen werden in einem einzigen, nicht aufteilbaren Schritt ausgeführt.

```
IF Object = FALSE THEN
  Object := TRUE;
  RETURN TRUE;
ELSE
  RETURN FALSE;
ENDIF
```

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TestAndSet`.

Beispiel 1:

```
LOCAL VAR bool doit_inuse := FALSE;
...
PROC doit(...)
  WaitUntil TestAndSet (doit_inuse);
  ...
  doit_inuse := FALSE;
ENDPROC
```

Wenn ein Modul als integriert (built-in) und zur gemeinsamen Nutzung (shared) installiert ist, kann eine lokale Modulvariable vor gleichzeitigem Zugriff aus verschiedenen Tasks schützen.

Hinweis

In diesem Fall mit installierten integrierten Modulen und beim Verwenden einer persistenten Variablen als Semaphor-Objekt: Wenn die Programmabarbeitung in der Routine `doit` gestoppt wird und der Programmzeiger zu `main` versetzt wird, wird die Variable `doit_inuse` nicht zurückgesetzt. Um dies zu vermeiden, sollte die Variable `doit_inuse` in der START-Ereignisroutine auf `FALSE` gesetzt werden.

Syntax

```
TestAndSet '('
  [ Object ':=' ] < variable or persistent (INOUT) of bool> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Warten, bis die Variable nicht gesetzt ist - und sie dann setzen (Typ "Warten mit Interrupt-Steuerung")	*WaitTestAndSet – Warten, bis Variable FALSE wird, dann einstellen auf Seite 1093*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.201 TestDI - Prüfen, ob ein digitaler Eingang gesetzt ist

Verwendung

`TestDI` wird verwendet, um zu prüfen, ob ein digitaler Eingang gesetzt ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TestDI`.

Beispiel 1:

```
IF TestDI (di2) THEN . . .
```

Wenn der aktuelle Wert des Signals `di2` gleich `1` ist, dann . . .

```
IF NOT TestDI (di2) THEN . . .
```

Wenn der aktuelle Wert des Signals `di2` gleich `0` ist, dann . . .

```
WaitUntil TestDI(di1) AND TestDI(di2);
```

Die Programmabarbeitung fährt erst fort, nachdem beide Eingänge `di1` und `di2` gesetzt wurden.

Rückgabewert

Datentyp: `bool`

`TRUE` = Der aktuelle Wert des Signals ist gleich 1.

`FALSE` = Der aktuelle Wert des Signals ist gleich 0.

Argumente

```
TestDI (Signal)
```

`Signal`

Datentyp: `signaldi`

Der Name des zu prüfenden Signals.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
`ERR_SIG_NOT_VALID`	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).

Syntax

```
TestDI '('
  [ Signal ':=' ] < variable (VAR) of signaldi > ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Ablesen eines digitalen Eingangssignalwerts	*signalxx - Digitale und analoge Signale auf Seite 1791*
Lesen des Werts eines digitalen Ausgangssignals	*DOutput - Lesen des aktuellen Werts von einem digitalen Ausgangssignal auf Seite 1295*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.202 TestSignRead - Lesen eines Prüfsignalwerts

Verwendung

`TestSignRead` wird verwendet, um den aktuellen Wert eines Prüfsignals zu lesen.

Diese Funktion gibt abhängig von der Kanalangabe in der Instruktion `TestSignDefine` den aktuellen Wert der neuesten Abtastungen zurück.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TestSignRead`.

Siehe auch *Weitere Beispiele auf Seite 1555*.

Beispiel 1:

```
CONST num speed_channel:=1;
VAR num speed_value;
...
TestSignDefine speed_channel, speed, orbit, 1, 0;
...
! During some movements with orbit’s axis 1
speed_value := TestSignRead(speed_channel);
...
TestSignReset;
```

`speed_value` wird der Mittelwert der letzten 8 Abtastungen (alle 0,5 ms) des Prüfsignals `speed` auf Kanal `speed_channel` zugewiesen, der als Kanal `1` definiert ist. Der Kanal `speed_channel` misst die Geschwindigkeit von `axis 1` an der mechanischen Einheit `orbit`.

Rückgabewert

Datentyp: `num`

Der numerische Wert in SI-Einheiten an der Motorseite für den angegebenen Kanal gemäß der Definition in der Instruktion `TestSignDefine`.

Argumente

```
TestSignRead (Channel)
```

`Channel`

Datentyp: `num`

Die Kanalnummern 1–12 des zu lesenden Prüfsignals. Dieselbe Nummer muss in der Definitionsinstruktion `TestSignDefine` angegeben werden.

Programmabarbeitung

Gibt den aktuellen Wert der neuesten Abtastungen abhängig von der Kanalangabe in der Instruktion `TestSignDefine` zurück.

Vordefinierte Prüfsignale mit gültigen SI-Einheiten für externe Manipulatorachsen siehe unter dem Datentyp `testsignal`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TestSignRead`.

Beispiel 1:

```
CONST num torque_channel:=2;
VAR num torque_value;
VAR intnum timer_int;
CONST jointtarget psync := [...];
...
PROC main()
  CONNECT timer_int WITH TorqueTrap;
  ITimer \Single, 0.05, timer_int;
  TestSignDefine torque_channel, torque_ref, IRBP_K, 2, 0.001;
  ...
  MoveAbsJ psync \NoEOffs, v5, fine, tool0;
  ...
  IDelete timer_int;
  TestSignReset;

TRAP TorqueTrap
  IF (TestSignRead(torque_channel) > 6) THEN
    TPWrite "Torque pos = " + ValToStr(CJointT());
    Stop;
  ELSE
    IDelete timer_int;
    CONNECT timer_int WITH TorqueTrap;
    ITimer \Single, 0.05, timer_int;
  ENDIF
ENDTRAP
```

Wenn der Drehmomentverweis auf der Motorseite für Achse `2` des Manipulators `IRBP_K` während der langsamen Bewegung zu Position `psync` das erste Mal größer als `6` Nm ist, wird die Achsenposition am FlexPendant angezeigt.

Syntax

```
TestSignRead '('
  [ Channel ':='] <expression (IN) of num> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Prüfsignal	*testsignal – Prüfsignal auf Seite 1821*
Definieren eines Prüfsignals	*TestSignDefine - Definieren eines Prüfsignals auf Seite 881*
Zurücksetzen eines Prüfsignals	*TestSignReset - Zurücksetzen aller Prüfsignaldefinitionen auf Seite 883*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.203 TextGet - Abrufen von Text aus Systemtexttabellen

Verwendung

Mit `TextGet` wird eine Zeichenfolge aus den Systemtexttabellen abgerufen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `TextGet`.

Beispiel 1:

```
VAR string text1;
...
text1 := TextGet(14, 5);
```

Der Variablen `text1` wird der Text aus der Textressource `14` und Index `5` zugewiesen.

Beispiel 2:

```
...
TPWrite TextGet(14, 511 \Arg1:="robot" \Arg2:="fast");
```

Der in Textressource `14` und Index `511` gespeicherte Text wird gelesen. Der aus den Texttabellen ausgelesene Text sieht folgendermaßen aus: `This is a {1} and this {1} is {2}`. Die in den optionalen Argumenten `Arg1` und `Arg2` verwendeten Zeichenfolgen ersetzen `{1}` und `{2}`. Die Ausgabe ist dann:

```
This is a robot and this robot is fast
```

Rückgabewert

Datentyp: `string`

Angegebener Text aus den Systemtexttabellen.

Argumente

```
TextGet ( Table Index \Arg1 \Arg2 \Arg3 \Arg4 \Arg5 \Arg6)
```

`Table`

Datentyp: `num`

Die Nummer der Texttabelle (positive Ganzzahl).

`Index`

Datentyp: `num`

Die Indexnummer (positive Ganzzahl) in der Texttabelle.

`[\Arg1]`

Datentyp: `string`

Wenn die aus der Texttabelle ausgelesene Zeichenfolge {1} enthält, dann wird {1} durch die Zeichenfolge in diesem Argument ersetzt.

`[\Arg2]`

Datentyp: `string`

Wenn die aus der Texttabelle ausgelesene Zeichenfolge {2} enthält, dann wird {2} durch die Zeichenfolge in diesem Argument ersetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\Arg3]

Datentyp: string

Wenn die aus der Texttabelle ausgelesene Zeichenfolge {3} enthält, dann wird {3} durch die Zeichenfolge in diesem Argument ersetzt.

[\Arg4]

Datentyp: string

Wenn die aus der Texttabelle ausgelesene Zeichenfolge {4} enthält, dann wird {4} durch die Zeichenfolge in diesem Argument ersetzt.

[\Arg5]

Datentyp: string

Wenn die aus der Texttabelle ausgelesene Zeichenfolge {5} enthält, dann wird {5} durch die Zeichenfolge in diesem Argument ersetzt.

[\Arg6]

Datentyp: string

Wenn die aus der Texttabelle ausgelesene Zeichenfolge {6} enthält, dann wird {6} durch die Zeichenfolge in diesem Argument ersetzt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_TXTNOEXIST	Tabelle oder Index ist nicht gültig und es kann keine Textzeichenfolge aus den Systemtexttabellen abgerufen werden.

Einschränkungen

Die gesamte Stringlänge in RAPID ist 80. Dies gilt für alle Eingaben als Ergebnis der Funktion.

Syntax

```
TextGet '('
  [Table ':='] <expression (IN) of num>','
  [Index ':='] <expression (IN) of num>
  ['\' Arg1 ':=' <expression (IN) of string>]
  ['\' Arg2 ':=' <expression (IN) of string>]
  ['\' Arg3 ':=' <expression (IN) of string>]
  ['\' Arg4 ':=' <expression (IN) of string>]
  ['\' Arg5 ':=' <expression (IN) of string>]
  ['\' Arg6 ':=' <expression (IN) of string>]')'
```

Eine Funktion mit einem Rückgabewert des Typs string.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.203 TextGet - Abrufen von Text aus Systemtexttabellen

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Abrufen der Texttabellennummer	*TextTabGet - Abrufen der Texttabellennummer auf Seite 1561*
Installieren einer Texttabelle	*TextTabInstall - Installieren einer Texttabelle auf Seite 884*
Format von Textdateien	*Technical reference manual - RAPID kernel*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.204 TextTabFreeToUse - Prüfen, ob Texttabelle frei ist

Verwendung

`TextTabFreeToUse` sollte verwendet werden, um zu prüfen, ob der Texttabellenname (text resource string) zur Verfügung steht (noch nicht im System installiert ist), d. h., ob es möglich ist, die Texttabelle im System zu installieren.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TextTabFreeToUse`.

Beispiel 1:

```
! System Module with Event Routine to be executed at event
! POWER ON, RESET or START

PROC install_text()
  IF TextTabFreeToUse("text_table_name") THEN
    TextTabInstall "HOME:/text_file.xml";
  ENDIF
ENDPROC
```

Wenn die Ereignisroutine `install_text` ausgeführt wird, gibt die Funktion `TextTabFreeToUse` den Wert `TRUE` zurück und die Textdatei `text_file.xml` wird im System installiert. Danach können die installierten Zeichenfolgen mit den Funktionen `TextTabGet` und `TextGet` vom System in RAPID abgerufen werden.

Bei der nächsten Abarbeitung der Ereignisroutine `install_text` gibt die Funktion `TextTabFreeToUse` den Wert `FALSE` zurück und die Installation wird **nicht** wiederholt.

Rückgabewert

Datentyp: `bool`

Diese Funktion gibt Folgendes zurück:

- `TRUE`, wenn die Texttabelle **noch nicht** im System installiert ist
- `FALSE`, wenn die Texttabelle bereits im System installiert ist

Argumente

```
TextTabFreeToUse ( TableName )
```

`TableName`

Datentyp: `string`

Name der Texttabelle (Zeichenfolge mit max. 20 Zeichen). Siehe `<text_resource>` in *Technical reference manual - RAPID kernel*, Abschnitt *Textdateien*. Der String `text_resource` ist der Name der Texttabelle.

Einschränkungen

Einschränkungen für die Installation von Texttabellen (Textquellen) im System:

- Es ist nicht möglich, dieselbe Texttabelle öfter als einmal im System zu installieren.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.204 TextTabFreeToUse - Prüfen, ob Texttabelle frei ist

RobotWare - OS

Fortsetzung

- Es ist nicht möglich, eine einzelne Texttabelle vom System zu deinstallieren (herauszunehmen). Der einzige Weg, Texttabellen vom System zu deinstallieren, ist ein Neustart der Steuerung mithilfe des Neustartmodus **System zurücksetzen**. Alle Texttabellen (System- und benutzerdefinierte Tabellen) werden dann deinstalliert.

Syntax

```
TextTabFreeToUse '('
  [TableName ':='] <expression (IN) of string>')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Installieren einer Texttabelle	*TextTabInstall - Installieren einer Texttabelle auf Seite 884*
Format von Textdateien	*Technical reference manual - RAPID kernel*
Abrufen der Texttabellennummer	*TextTabGet - Abrufen der Texttabellennummer auf Seite 1561*
Abrufen von Text aus Systemtexttabellen	*TextGet - Abrufen von Text aus Systemtexttabellen auf Seite 1556*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.205 TextTabGet - Abrufen der Texttabellennummer

Verwendung

`TextTabGet` wird verwendet, um die Nummer einer anwenderdefinierten Texttabelle während der Ausführung abzurufen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `TextTabGet`.

In beiden Beispielen wird eine neue Texttabelle mit dem Namen `deburr_part1` für anwenderdefinierte Texte verwendet. Die neue Texttabelle hat den Dateiname n`deburr.xml`. Das Format .eng kann ebenfalls verwendet werden, allerdings wird .xml empfohlen.

Beispielcode im .xml-Format

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Resource Language="en" Name="deburr_part1">
  <Text Name="1">
    <Value>Part 1 is not in pos</Value>
    <Comment>Maximum length 80 chars</Comment>
  </Text>
  <Text Name="2">
    <Value>Identity of worked part: XYZ</Value>
    <Comment>Maximum length 80 chars</Comment>
  </Text>
  <Text Name="3">
    <Value>Part error in line 1</Value>
    <Comment>Maximum length 80 chars</Comment>
  </Text>
</Resource>
```

Beispiel 1:

```
VAR num text_res_no;
...
text_res_no := TextTabGet("deburr_part1");
```

Der Variablen `text_res_no` wird die Texttabellennummer für die definierte Texttabelle `deburr_part1` zugewiesen.

Beispiel 2:

```
ErrWrite TextGet(text_res_no, 1), TextGet(text_res_no, 2);
```

Eine Meldung wird im Roboterprotokoll gespeichert. Die Meldung wird auch am FlexPendant ausgegeben. Die Meldungen werden der Texttabelle `deburr_part1` entnommen:

Part 1 is not in pos

Identity of worked part: XYZ

Beispielcode im .eng-Format

```
# deburr.eng - USERS deburr_part1 english text description file
#
# DESCRIPTION:
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.205 TextTabGet - Abrufen der Texttabellennummer

RobotWare - OS

Fortsetzung

```
# Users text file for RAPID development
#
deburr_part1::
0:
RAPID S4: Users text table deburring part1
1:
Part 1 is not in pos
2:
Identity of worked part: XYZ
3:
Part error in line 1
#
# End of file
```

Rückgabewert

Datentyp: `num`

Die Texttabellennummer der definierten Texttabelle.

Argumente

```
TextTabGet ( TableName )
```

`TableName`

Datentyp: `string`

Name der Texttabelle.

Syntax

```
TextTabGet '('
  [TableName '='] <expression (IN) of string>';)'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Abrufen von Text aus Systemtexttabellen	*TextGet - Abrufen von Text aus Systemtexttabellen auf Seite 1556*
Installieren einer Texttabelle	*TextTabInstall - Installieren einer Texttabelle auf Seite 884*
Format von Textdateien	*Technical reference manual - RAPID kernel*
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.206 TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist

Verwendung

Die `TriggDataValid`-Funktion dient zur Überprüfung, ob eine `triggdata`-Variable gültig ist. Eine gültige `triggdata`-Variable ist eine Variable, die zuvor im Programm in einer der Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` verwendet wurde, um Triggerbedingungen und Triggeraktivitäten anzugeben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `TriggDataValid`.

Beispiel 1:

```
VAR triggdata trigg_array{25};
...
PROC MyTriggProcL(robtarget myrobt, \VAR triggdata T1 \VAR triggdata
     T2 \VAR triggdata T3)
  VAR num triggcnt:=2;
  ! Reset entire trigg_array array before using it
  FOR i FROM 1 TO 25 DO
    TriggDataReset trigg_array{i};
  ENDFOR
  TriggEquip trigg_array{1}, 10 \Start, 0 \DOp:=do1, SetValue:=1;
  TriggEquip trigg_array{2}, 40 \Start, 0 \DOp:=do2, SetValue:=1;
  ! Check if optional argument is present,
  ! and if any trigger condition has been setup in T1
  IF Present(T1) AND TriggDataValid(T1) THEN
    ! Copy actual trigger condition to trigg_array
    TriggDataCopy trigg_array{triggcnt}, T1;
    Incr triggcnt;
  ENDIF
  IF Present(T2) AND TriggDataValid(T2) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T2;
  ENDIF
  IF Present(T3) AND TriggDataValid(T3) THEN
    Incr triggcnt;
    TriggDataCopy trigg_array{triggcnt}, T3;
  ENDIF
  TriggL p1, v500, trigg_array, z30, tool2;
  ...
```

Das Verfahren `MyTriggProcL` oben nutzt die `TriggDataValid`-Instruktion, um zu prüfen, ob in den optionalen Argumenten `T1`, `T2` und `T3` irgendwelche gültigen Daten verwendet werden.

Rückgabewert

Datentyp: `bool`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.206 TriggDataValid - Prüfen, ob der Inhalt in einer triggdata-Variablen gültig ist

RobotWare - OS

Fortsetzung

Diese Funktion gibt Folgendes zurück:

- `TRUE`, wenn die Variable gültig ist, d.h. eine der Instruktionen `TriggIO`, `TriggEquip`, `TriggInt`, `TriggSpeed`, `TriggCheckIO` oder `TriggRampAO` verwendet wurde , um Triggerbedingungen und Triggeraktivitäten anzugeben.
- `FALSE`, wenn die Variable nicht verwendet wurde, um Triggerbedingungen und Triggeraktivitäten anzugeben.

Argumente

```
TriggDataValid TriggData
```

`TriggData`

Datentyp: `triggdata`

Die `triggdata`-Variable zur Überprüfung der Gültigkeit.

Syntax

```
TriggDataValid
  [TriggData ':=' ] < variable (VAR) of triggdata > ';'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Lineare Bewegung mit Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950*
Achsenbewegung mit Triggern	*TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Kreisförmige Bewegung mit Triggern	*TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung auf Seite 983*
Handhabung `triggdata`	*triggdata – Positioniereereignisse, trigg auf Seite 1833* *TriggDataReset - Zurücksetzen des Inhalts in einer triggdata-Variablen auf Seite 921* *TriggDataCopy - Kopieren Sie den Inhalt in eine triggdata-Variable auf Seite 919*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.207 Trunc - Schneidet einen numerischen Wert ab

Verwendung

`Trunc` (*Truncate*) wird verwendet, um einen numerischen Wert auf eine angegebene Anzahl von Dezimalstellen oder eine Ganzzahl abzuschneiden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `Trunc`.

Beispiel 1:

```
VAR num val;
val := Trunc(0.3852138\Dec:=3);
```

Die Variable `val` erhält den Wert `0.385`.

Beispiel 2:

```
reg1 := 0.3852138;
val := Trunc(reg1\Dec:=1);
```

Die Variable `val` erhält den Wert `0.3`.

Beispiel 3

```
val := Trunc(0.3852138);
```

Die Variable `val` erhält den Wert `0`.

Beispiel 4:

```
val := Trunc(0.3852138\Dec:=6);
```

Die Variable `val` erhält den Wert `0.385213`.

Rückgabewert

Datentyp: `num`

Der numerische Wert, auf die angegebene Anzahl an Dezimalstellen abgeschnitten.

Argumente

```
Trunc ( Val [\Dec] )
```

`Val`

Value

Datentyp: `num`

Der abzuschneidende numerische Wert.

`[\Dec]`

Decimals

Datentyp: `num`

Anzahl an Dezimalstellen.

Wenn die angegebene Anzahl an Dezimalstellen 0 ist oder das Argument weggelassen wird, wird der Wert auf eine Ganzzahl abgeschnitten.

Die Anzahl an Dezimalstellen darf nicht als negativer Wert oder mit größerer Genauigkeit als die verfügbare Genauigkeit für numerische Werte angegeben werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.207 Trunc - Schneidet einen numerischen Wert ab

RobotWare - OS

Fortsetzung

Maximale Anzahl an verwendbaren Dezimalzahlen ist 6.

Syntax

```
Trunc '('
  [ Val ':=' ] <expression (IN) of num>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Runden eines Werts	*Round - Rundet einen numerischen Wert auf Seite 1484*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.208 TruncDnum - Schneidet einen numerischen Wert ab

Verwendung

`TruncDnum` (*Truncate dnum*) wird verwendet, um einen numerischen Wert auf eine angegebene Anzahl von Dezimalstellen oder eine Ganzzahl abzuschneiden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `TruncDnum`.

Beispiel 1:

```
VAR dnum val;
val := TruncDnum(0.3852138754655357\Dec:=3);
```

Die Variable `val` erhält den Wert `0.385`.

Beispiel 2:

```
val := TruncDnum(0.3852138754655357\Dec:=1);
```

Die Variable `val` erhält den Wert `0.3`.

Beispiel 3

```
val := TruncDnum(0.3852138754655357);
```

Die Variable `val` erhält den Wert `0`.

Beispiel 4:

```
val := TruncDnum(0.3852138754655357\Dec:=15);
```

Die Variable `val` erhält den Wert `0.385213875465535`.

Beispiel 5:

```
val := TruncDnum(1000.3852138754655357\Dec:=15);
```

Die Variable `val` erhält den Wert `1000.38521387547`.

Rückgabewert

Datentyp: `dnum`

Der numerische Wert, auf die angegebene Anzahl an Dezimalstellen abgeschnitten.

Argumente

```
TruncDnum ( Val [\Dec] )
```

`Val`

Value

Datentyp: `dnum`

Der abzuschneidende numerische Wert.

`[\Dec]`

Decimals

Datentyp: `num`

Anzahl an Dezimalstellen.

Wenn die angegebene Anzahl an Dezimalstellen 0 ist oder das Argument weggelassen wird, wird der Wert auf eine Ganzzahl abgeschnitten.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.208 TruncDnum - Schneidet einen numerischen Wert ab

RobotWare - OS

Fortsetzung

Die Anzahl an Dezimalstellen darf nicht als negativer Wert oder mit größerer Genauigkeit als die verfügbare Genauigkeit für numerische Werte angegeben werden.

Maximale Anzahl an verwendbaren Dezimalzahlen ist 15.

Syntax

```
TruncDnum '('
  [ Val ':=' ] <expression (IN) of dnum>
  [ \Dec ':=' <expression (IN) of num> ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Abschneiden eines Werts	*Trunc - Schneidet einen numerischen Wert ab auf Seite 1565*
Runden eines Werts	*Round - Rundet einen numerischen Wert auf Seite 1484*
Runden eines Werts	*RoundDnum - Rundet einen numerischen Wert auf Seite 1486*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.209 Type - Abrufen des Datentypnamens für eine Variable.

Verwendung

Mit `Type` wird der Datentypname für die angegebene Variable im Argument `Data` abgerufen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `Type`.

Beispiel 1:

```
VAR string rettype;
VAR intnum intnumtype;
...
PROC main()
  rettype := Type(intnumtype);
  TPWrite "Data type name: " + rettype;
```

Die Druckausgabe lautet: "Data type name: intnum"

Beispiel 2:

```
VAR string rettype;
VAR intnum intnumtype;
...
PROC main()
  rettype := Type(intnumtype \BaseName);
  TPWrite "Data type name: " + rettype;
```

Die Druckausgabe lautet: "Data type name: num"

Beispiel 3

```
VAR string rettype;
VAR num numtype;
...
PROC main()
  rettype := Type(numtype);
  TPWrite "Data type name: " + rettype;
```

Die Druckausgabe lautet: "Data type name: num"

Rückgabewert

Datentyp: `string`

Eine Zeichenfolge mit dem Datentypnamen für die angegebene Variable im Argument `Data`.

Argumente

```
Type (Data [\BaseName])
```

`Data`

Data object name

Datentyp: `anytype`

Der Name der Variablen, für die der Datentypname abgerufen werden soll.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.209 Type - Abrufen des Datentypnamens für eine Variable.

RobotWare - OS
Fortsetzung

`[\BaseName]`

Base data type Name

Datentyp: `switch`

Die Funktion gibt den zugrunde liegenden Datentypnamen zurück,wenn die angegebenen `Data` einen deklarierten `ALIAS`-Datentyp aufweisen.

Syntax

```
Type '('
  [ Data ':=' ] < reference (REF) of anytype >
  [ '\' BaseName ] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Definition der Alias-Typen	*Technical reference manual - RAPID kernel* *ALIAS – Zuweisen eines Alias-Datentyps auf Seite 1637*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.210 UIAlphaEntry - Benutzer-Alpha-Eintrag

Verwendung

`UIAlphaEntry` (*User Interaction Alpha Entry*) ermöglicht dem Bediener die Eingabe einer Zeichenfolge auf dem verfügbaren Benutzergerät, z.B. dem FlexPendant. An den Bediener wird eine Meldung ausgegeben, und dieser antwortet mit einer Zeichenfolge. Die Zeichenfolge wird dann zurück an das Programm übertragen.

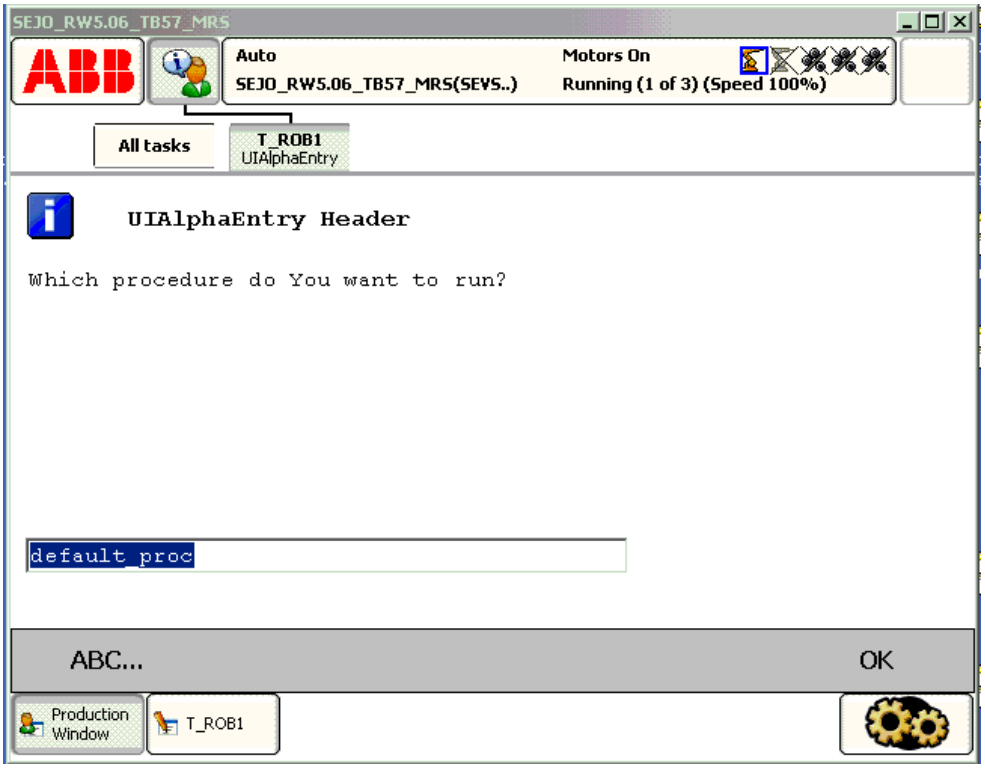
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIAlphaEntry`.

Siehe *Weitere Beispiele auf Seite 1576*.

Beispiel 1:

```
VAR string answer;
...
answer := UIAlphaEntry(
  \Header:= "UIAlphaEntry Header",
  \Message:= "Which procedure do You want to run?"
  \Icon:=iconInfo
  \InitString:= "default_proc");
%answer%;
```



xx0500002437

Das Meldungsfenster wird mit Icon, Header, Meldung und Anfangszeichenfolge auf FlexPendant angezeigt. Der Benutzer kann die Anfangszeichenfolge bearbeiten oder über Soft-Keyboard eine neue Zeichenfolge schreiben. Die Programmabarbeitung wartet, bis auf OK getippt wird. Dann wird die Zeichenfolge

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

in der Variablen `answer` zurückgegeben. Anschließend ruft das Programm die angegebene Prozedur mit später Bindung auf.

Rückgabewert

Datentyp: `string`

Diese Funktion gibt die Eingangszeichenfolge zurück.

Brechen der Funktion über `\BreakFlag::`

- Wenn der Parameter `\InitString` angegeben ist, wird diese Zeichenfolge zurückgegeben.
- Wenn der Parameter `\InitString` nicht angegeben ist, wird eine leere Zeichenfolge `""` zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine abgebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UIAlphaEntry ([\Header] [\Message]|[\MsgArray]
     [\Wrap][\Icon][\InitString] [\MaxTime] [\DIBreak] [\DIPassive]
     [\DOBreak] [\DOPassive] [\PersBoolBreak] [\PersBoolPassive]
     [\BreakFlag] [\UIActiveSignal])
```

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`[\Message]`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Max. 55 Zeichen.

`[\MsgArray]`

Message Array

Datentyp: `string`

Several text lines from an array to be written on the display.

Es kann jeweils nur ein Parameter `\Message` oder `\MsgArray` verwendet werden.

Der maximal verfügbare Platz beträgt 9 Zeilen mit jeweils 55 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile am Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Definiert das anzuzeigende Icon. Only one of the predefined icons of type icondata can be used. Siehe *Vordefinierte Daten auf Seite 1575*.

Standardmäßig wird kein Icon angezeigt.

`[\InitString]`

Datentyp: `string`

Standardmäßig wird eine Anfangszeichenfolge im Texteingabefeld am Display angezeigt.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn die OK-Taste innerhalb dieser Zeitdauer nicht betätigt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOBreak]`

Digital Output Break

Datentyp: `signaldo`

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOPassive]`

Digital Output Passive

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.210 UIAlphaEntry - Benutzer-Alpha-Eintrag
RobotWare-OS
Fortsetzung

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DOBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DOBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolBreak]`

Persistent Boolean Break

Datentyp: `bool`

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolPassive]`

Persistent Boolean Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `PersBoolBreak` verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn die persistente boolesche Variable `PersBoolBreak` auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\BreakFlag]`

Datentyp: `errnum`

Eine Variable, die den Fehlercode enthält, wenn `MaxTime`, `DIBreak`, `DOBreak` oder `PersBoolBreak` verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK`, `ERR_TP_DOBREAK` und `ERR_TP_PERSBOOLBREAK` können verwendet werden, um die Ursache zu bestimmen.

`[\UIActiveSignal]`

Datentyp: `signaldo`

Das digitale Ausgangssignal, das im optionalen Argument `UIActiveSignal` verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Programmabarbeitung

Das Alpha-Meldungsfenster mit Alpha Pad, Icon, Header, Meldungszeilen und Anfangszeichenfolge wird den programmierten Argumenten entsprechend angezeigt. Die Programmabarbeitung wartet, bis der Benutzer die Zeichenfolge bearbeitet oder eine neue Zeichenfolge erstellt hat und dann die OK-Taste betätigt hat oder das Meldungsfenster durch ein Timeout oder eine Signalaktion unterbrochen wird. Die Anfangszeichenfolge und der Grund der Unterbrechung werden zurück in das Programm übertragen.

Neue Meldungsfenster auf Interrupt-Ebene werden von Meldungsfenstern auf Basisebene beachtet.

Vordefinierte Daten

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.

Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak)), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIAlphaEntry`.

Beispiel 1:

```
VAR errnum err_var;
VAR string answer;
VAR string logfile;
...
answer := UIAlphaEntry (\Header:= "Log file name:"
  \Message:= "Enter the name of the log file to create?"
  \Icon:=iconInfo
  \InitString:= "signal.log"
  \MaxTime:=60
  \DIBreak:=di5\BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
    logfile:="signal.log";
  CASE 0:
    ! Operator answer
    logfile := answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt und der Bediener kann eine Zeichenfolge eingeben und OK drücken. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Im Programm ist die Ursache ersichtlich und es können entsprechende Maßnahmen eingeleitet werden.

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `UIAlphaEntry` häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
UIAlphaEntry '('
  ['\' Header ':=' <expression (IN) of string>]
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':='<array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' InitString ':='<expression (IN) of string>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':='<variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
['\' PersBoolPassive]
['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
Prozeduraufruf mit später Bindung	*Technisches Referenzhandbuch - RAPID Overview*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.211 UIClientExist - Vorhandensein eines Benutzer-Clients

Verwendung

`UIClientExist` (*User Interaction Client Exist*) wird verwendet, um zu überprüfen, ob ein Benutzergerät wie das FlexPendant mit der Steuerung verbunden ist.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIClientExist`.

Beispiel 1:

```
IF UIClientExist() THEN
  ! Possible to get answer from the operator
  ! The TPReadFK and UIMsgBox ... can be used
ELSE
  ! Not possible to communicate with any operator
ENDIF
```

Es wird überprüft, ob es möglich ist, eine Antwort vom Bediener des Systems zu erhalten.

Rückgabewert

Datentyp: `bool`

Gibt `TRUE`, wenn ein FlexPendant mit dem System verbunden ist, andernfalls `FALSE`.

Einschränkungen

`UIClientExist` gibt für bis zu 16 Sekunden `TRUE` zurück. Anschließend wird das FlexPendant entfernt. Nach Ablauf dieser Zeit gibt `UIClientExist` den Wert `FALSE` zurück (wenn eine unterbrochene Netzwerkverbindung des FlexPendant festgestellt wird). Dieselbe Einschränkung gilt, wenn das FlexPendant wieder verbunden wird.

Syntax

```
UIClientExist '(' ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.212 UIDnumEntry - Eintrag eines numerischen Werts durch den Benutzer

Verwendung

`UIDnumEntry` (*User Interaction Number Entry*) ermöglicht dem Bediener die Eingabe eines numerischen Wertes auf dem verfügbaren Benutzergerät, z. B. dem FlexPendant. An den Bediener wird eine Meldung ausgegeben, und dieser antwortet mit einem numerischen Wert. Anschließend wird der numerische Wert überprüft, bestätigt und zurück an das Programm übertragen.

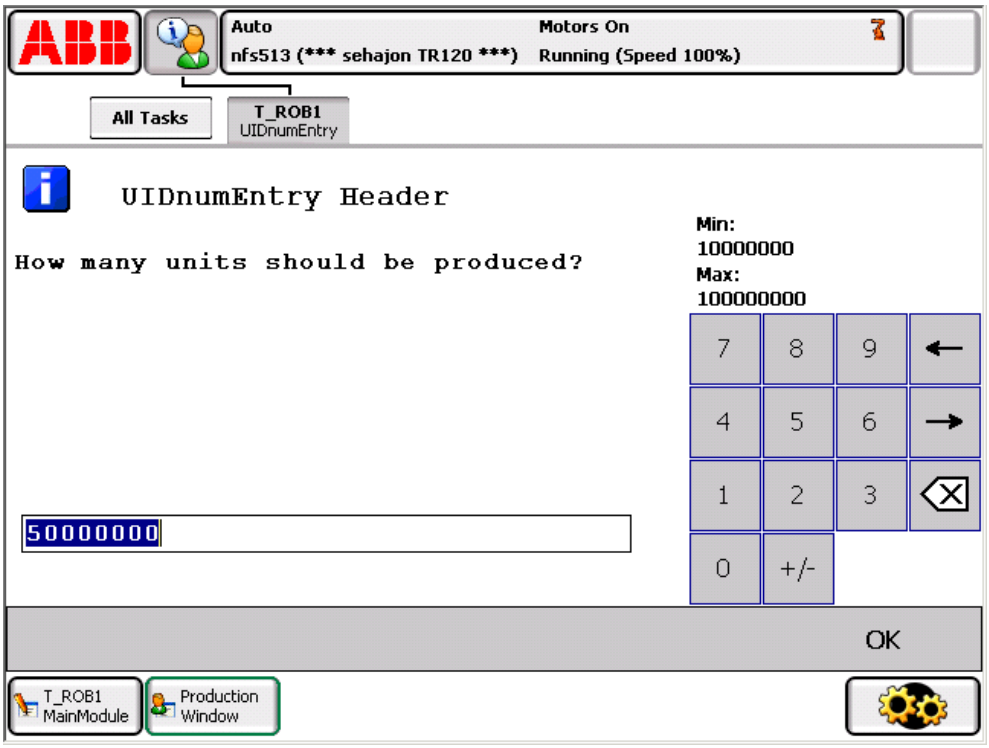
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIDnumEntry`.

Siehe auch *Weitere Beispiele auf Seite 1585*.

Beispiel 1:

```
VAR dnum answer;
...
answer := UIDnumEntry(
  \Header:="UIDnumEntry Header"
  \Message:="How many units should be produced?"
  \Icon:=iconInfo
  \InitValue:=50000000
  \MinValue:=10000000
  \MaxValue:=100000000
  \AsInteger);
```



xx0900001064

Das numerische Meldungsfenster wird mit Icon, Header, Meldung, Anfangs-, Maximal- und Minimalwert auf FlexPendant angezeigt. Im Meldungsfenster wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

überprüft, ob der Bediener eine Ganzzahl im Wertebereich ausgewählt hat. Die Programmabarbeitung wartet, bis OK gedrückt wurde und dann der gewählte numerische Wert zurückgegeben wird.

Rückgabewert

Datentyp: `dnum`

Diese Funktion gibt den numerischen Eingangswert zurück.

Brechen der Funktion über `\BreakFlag`:

- Wenn der Parameter `\InitValue` angegeben ist, wird dieser Wert zurückgegeben.
- Wenn der Parameter `\InitValue` nicht angegeben ist, wird der Wert 0 zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine gebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UIDnumEntry ( [\Header] [\Message] | [\MsgArray]
     [\Wrap][\Icon][\InitValue] [\MinValue] [\MaxValue]
     [\AsInteger][\MaxTime] [\DIBreak] [\DIPassive] [\DOBreak]
     [\DOPassive] [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
     [\UIActiveSignal])
```

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`[\Message]`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Maximal 40 Zeichen.

`[\MsgArray]`

Message Array

Datentyp: `string`

Several text lines from an array to be written on the display.

Es kann jeweils nur der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der maximal verfügbare Platz beträgt 9 Zeilen mit jeweils 40 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile am Display angezeigt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.212 UIDnumEntry - Eintrag eines numerischen Werts durch den Benutzer
RobotWare - OS
Fortsetzung

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1584*.

Standardmäßig wird kein Icon angezeigt.

`[\InitValue]`

Datentyp: `dnum`

Der Anfangswert, der im Eingabefenster angezeigt wird.

`[\MinValue]`

Datentyp: `dnum`

Der Mindestwert für den Rückgabewert.

`[\MaxValue]`

Datentyp: `dnum`

Der Maximalwert für den Rückgabewert.

`[\AsInteger]`

Datentyp: `switch`

Entfernt den Dezimalpunkt vom Nummernblock, um sicherzustellen, dass der Rückgabewert eine Ganzzahl ist.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn die OK-Taste innerhalb dieser Zeitdauer nicht betätigt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

(oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DOBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolBreak]

Persistent Boolean Break

Datentyp: bool

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolPassive]

Persistent Boolean Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument PersBoolBreak verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn die persistente boolesche Variable PersBoolBreak auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignal verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Das numerische Meldungsfenster wird mit Nummernblock, Icon, Header, Meldungszeilen, Anfangs-, Maximal- und Minimalwert den programmierten Argumenten entsprechend angezeigt. Die Programmabarbeitung wartet, bis der Benutzer einen zulässigen numerischen Wert eingibt und dann die OK-Taste betätigt oder das Meldungsfenster durch ein Timeout oder eine Signalaktion unterbrochen wird. Der numerische Wert und der Grund der Unterbrechung werden in das Programm zurückübertragen.

Neue Meldungsfenster auf Interrupt-Ebene werden von Meldungsfenstern auf Basisebene beachtet.

Vordefinierte Daten

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.
ERR_UI_INITVALUE	Der Anfangswert (Parameter \InitValue) liegt nicht zwischen dem Minimal- und Maximalwert (Parameter \MinValue und \MaxValue).
ERR_UI_MAXMIN	Der Mindestwert (Parameter \MinValue) ist größer als der Maximalwert (Parameter \MaxValue).
ERR_UI_NOTINT	Der Anfangswert (Parameter \InitValue) ist keine Ganzzahl, wie im Parameter \AsInteger festgelegt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Diese Situation kann nur in der Fehlerbehandlungsroutine behoben werden.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion UIDnumEntry.

Beispiel 1:

```
VAR errnum err_var;
VAR dnum answer;
VAR dnum distance;
...
answer := UIDnumEntry (\Header:= "BWD move on path"
  \Message:="Enter the path overlap?" \Icon:=iconInfo
  \InitValue:=5 \MinValue:=0 \MaxValue:=10
  \MaxTime:=60 \DIBreak:=di5 \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer distance := 5;
  CASE 0
    ! Operator answer
    distance := answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt und der Bediener kann einen numerischen Wert eingeben und OK drücken. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Im Programm ist die Ursache ersichtlich und es können entsprechende Maßnahmen eingeleitet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.212 UIDnumEntry - Eintrag eines numerischen Werts durch den Benutzer

RobotWare - OS

Fortsetzung

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `UIDnumEntry` häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
UIDnumEntry '('
  ['\' Header ':=' <expression (IN) of string>]
  [Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' InitValue ':=' <expression (IN) of dnum>]
  ['\' MinValue ':=' <expression (IN) of dnum>]
  ['\' MaxValue ':=' <expression (IN) of dnum>]
  ['\' AsInteger]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\' PersBoolPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Zahlensynchronisierung“	*UIDnumTune - Benutzer-Zahlensynchronisierung auf Seite 1588*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.213 UIDnumTune - Benutzer-Zahlensynchronisierung

Verwendung

`UIDnumTune` (*User Interaction Number Tune*) ermöglicht dem Bediener das Abstimmen eines numerischen Wertes auf dem verfügbaren Benutzergerät, z.B. dem FlexPendant. An den Bediener wird eine Meldung ausgegeben, und dieser stimmt einen numerischen Wert ab. Anschließend wird der abgestimmte numerische Wert überprüft, bestätigt und zurück an das Programm übertragen.

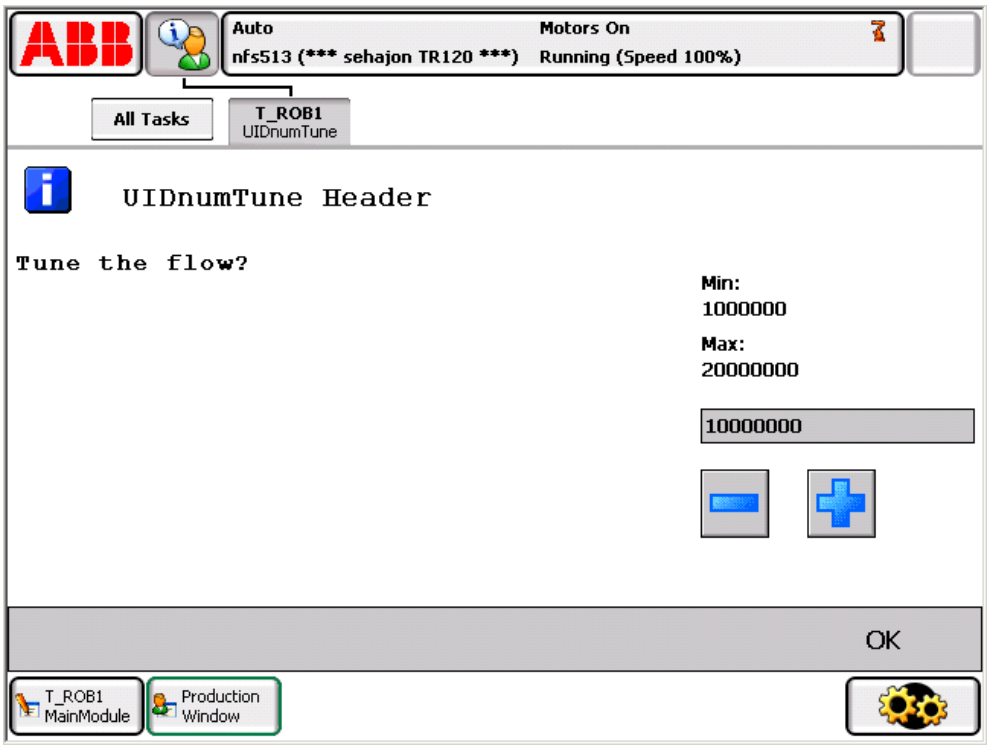
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIDnumTune`.

Siehe auch *Weitere Beispiele auf Seite 1593*.

Beispiel 1:

```
VAR dnum flow;
...
flow := UIDnumTune(
  \Header:="UIDnumTune Header"
  \Message:="Tune the flow?"
  \Icon:=iconInfo,
  10000000,
  1000000
  \MinValue:=1000000
  \MaxValue:=20000000);
```



xx0900001063

Das Abstimmungs-Meldungsfenster wird mit Icon Header, Meldung, Anfangs-, Inkremental-, Maximal- und Minimalwert auf FlexPendant angezeigt. Im

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Meldungsfenster wird überprüft, ob der Bediener den `flow`-Wert mit der Schrittgröße 1000000 vom Anfangswert 10000000 und im Wertebereich 1000000-20000000 abgestimmt hat. Die Programmabarbeitung wartet, bis auf OK getippt wird und dann der gewählte numerische Wert zurückgegeben und in der Variablen `flow` gespeichert wird.

Rückgabewert

Datentyp: `dnum`

Diese Funktion gibt den abgestimmten numerischen Wert zurück.

Wenn die Funktion über ein `\BreakFlag` gebrochen wird, wird der angegebene `InitValue` zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine gebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UIDnumTune ( [\Header] [\Message] | [\MsgArray] [\Wrap]
     [\Icon]InitValue Increment [\MinValue] [\MaxValue]
     [\MaxTime][\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive]
     [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
     [\UIActiveSignal])
```

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`[\Message]`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Maximal 40 Zeichen.

`[\MsgArray]`

Message Array

Datentyp: `string`

Several text lines from an array to be written on the display.

Es kann jeweils nur der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der maximal verfügbare Platz beträgt 11 Zeilen mit jeweils 40 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile am Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Definiert das anzuzeigende Icon. Only one of the predefined icons of type icondata can be used. Siehe *Vordefinierte Daten auf Seite 1592*.

Standardmäßig wird kein Icon angezeigt.

`InitValue`

Initial Value

Datentyp: `dnum`

Der Anfangswert, der im Eingabefenster angezeigt wird.

`Increment`

Datentyp: `dnum`

Dieser Parameter gibt an, um wie viel sich der Wert ändern soll, wenn die Plus- oder Minus-Schaltfläche betätigt wird.

`[\MinValue]`

Datentyp: `dnum`

Der Mindestwert für den Rückgabewert.

`[\MaxValue]`

Datentyp: `dnum`

Der Maximalwert für den Rückgabewert.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn die OK-Taste innerhalb dieser Zeitdauer nicht betätigt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

(oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mit der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DOBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolBreak]

Persistent Boolean Break

Datentyp: bool

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolPassive]

Persistent Boolean Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument PersBoolBreak verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn die persistente boolesche Variable PersBoolBreak auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignal verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Das numerische Abstimmungs-Meldungsfenster mit den Abstimmungsschaltflächen +/-, Icon, Header, Meldungszeilen, Anfangs-, Inkremental-, Maximal- und Minimalwert wird den programmierten Argumenten entsprechend angezeigt. Die Programmabarbeitung wartet, bis der Benutzer den numerischen Wert abgestimmt hat und dann die OK-Taste betätigt oder das Meldungsfenster durch ein Timeout oder eine Signalaktion unterbrochen wird. Der numerische Wert und der Grund der Unterbrechung werden in das Programm zurückübertragen.

Neue Meldungsfenster auf TRAP-Ebene erhalten den Fokus von Meldungsfenstern auf der Basisebene.

Vordefinierte Daten

```
!Icons:
CONST icondata iconNone := 0;
CONST icondata iconInfo := 1;
CONST icondata iconWarning := 2;
CONST icondata iconError := 3;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.
ERR_UI_INITVALUE	Der Anfangswert (Parameter \InitValue) liegt nicht zwischen dem Minimal- und Maximalwert (Parameter \MinValue und \MaxValue).
ERR_UI_MAXMIN	Der Mindestwert (Parameter \MinValue) ist größer als der Maximalwert (Parameter \MaxValue).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion UIDnumTune.

Beispiel 1:

```
VAR errnum err_var;
VAR dnum tune_answer;
VAR dnum distance;
...
tune_answer := UIDnumTune (\Header:=" BWD move on path"
  \Message:="Enter the path overlap?" \Icon:=iconInfo,
  5, 1 \MinValue:=0 \MaxValue:=10
  \MaxTime:=60 \DIBreak:=di5 \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
    distance := 5;
  CASE 0:
    ! Operator answer
    distance := tune_answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt und der Bediener kann einen numerischen Wert abstimmen und OK drücken. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Im Programm ist die Ursache ersichtlich und es können entsprechende Maßnahmen eingeleitet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.213 UIDnumTune - Benutzer-Zahlensynchronisierung

RobotWare - OS

Fortsetzung

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `UIDnumTune` häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
UIDnumTune '('
  ['\' Header ':=' <expression (IN) of string>]
  ['\' Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>] ',']
  [InitValue ':=' ] <expression (IN) of dnum> ','
  [Increment ':=' ] <expression (IN) of dnum>
  ['\' MinValue ':=' <expression (IN) of dnum>]
  ['\' MaxValue ':=' <expression(IN) of dnum>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\' PersBoolPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `dnum`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahleneingabe“	*UIDnumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1580*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.214 UIListView - Anzeigen der Benutzerliste

Verwendung

`UIListView` (*User Interaction List View*) wird zur Definition von Menülisten mit Text und bei Bedarf auch mit Icons verwendet, die auf einem Benutzergerät wie dem FlexPendant verfügbar sind. Das Menü ist in zwei unterschiedlichen Varianten verfügbar, einer mit Validierungsschaltflächen und einer, die sofort auf die Benutzerauswahl reagiert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIListView`.

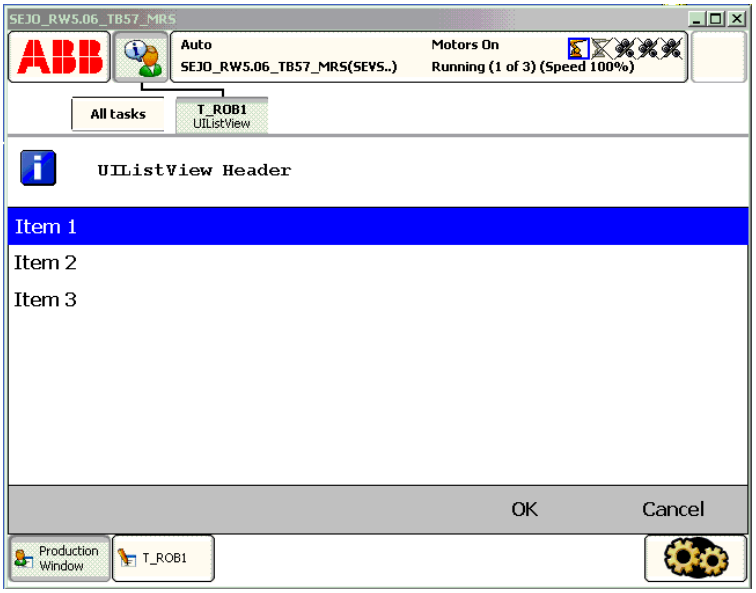
Siehe auch *Weitere Beispiele auf Seite 1602*.

Beispiel 1:

```
CONST listitem list{3} := [ ["","Item 1"], ["","Item 2"],
     ["","Item3"] ];
VAR num list_item;
VAR btnres button_answer;
...
list_item := UIListView (
  \Result:=button_answer
  \Header:="UIListView Header",
  list
  \Buttons:=btnOKCancel
  \Icon:=iconInfo
  \DefaultIndex:=1);
IF button_answer = resOK THEN
  IF list_item = 1 THEN
    ! Do item1
  ELSEIF list_item = 2 THEN
    ! Do item 2
  ELSE
    ! Do item3
  ENDIF
ELSE
  ! User has select Cancel
ENDIF
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.



xx0500002416

Die Menüliste wird mit Icon, Header, Menü `Item 1 ... Item 3` und den Schaltflächen auf FlexPendant angezeigt. Die Programmabarbeitung wartet, bis „OK“ oder „Abbrechen“ gedrückt wird. Sowohl die Auswahl in der Liste als auch die betätigte Schaltfläche werden an das Programm übertragen.

Rückgabewert

Datentyp: `num`

Diese Funktion gibt die Benutzerauswahl im Listenmenü entsprechend dem Index in dem Datenfeld an, der im Parameter `ListItems` angegeben wird.

Brechen der Funktion über `\BreakFlag`:

- Wenn der Parameter `\DefaultIndex` angegeben ist, wird dieser Index zurückgegeben.
- Wenn der Parameter `\DefaultIndex` nicht angegeben ist, wird 0 zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine gebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UIListView ( [\Result] [\Header] ListItems [\Buttons] | [\BtnArray]
     [\Icon] [\DefaultIndex ] [\MaxTime] [\DIBreak] [\DIPassive]
     [\DOBreak] [\DOPassive] [\PersBoolBreak] [\PersBoolPassive]
     [\BreakFlag] [\UIActiveSignal])
```

`[\Result]`

Datentyp: `btnres`

Der numerische Wert der Schaltfläche, die aus dem Listenmenüfeld ausgewählt wurde.

Wenn das Argument `\Buttons` verwendet wird, werden die vordefinierten symbolischen Konstanten des Typs `btnres` zurückgegeben. Wenn das Argument

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`\BtnArray` verwendet wird, wird der entsprechende Datenfeldindex zurückgegeben.

Das Argument `\Result` ergibt 0, wenn es auf resUnkwn gesetzt wird und eine der folgenden Bedingungen erfüllt ist:

- Weder der Parameter `\Buttons` noch der Parameter `\BtnArray` wird verwendet,
- Das Argument `\Buttons:=btnNone` wird verwendet
- Brechen der Funktion über `\BreakFlag` oder die `ERROR`-Behandlungsroutine

Siehe *Vordefinierte Daten auf Seite 1601*.

`[\Header]`

Datentyp: `string`

Der zu schreibende Header-Text oben im Listenmenüfenster. Max. 40 Zeichen.

`ListItems`

Datentyp: `listitem`

Ein Datenfeld, aus dem mindestens ein Listenmenüobjekt angezeigt werden soll, das aus Folgendem besteht:

Komponente `image` des Typs `string`:

Der Name des zu verwendenden Icon-Bildes. Um eigene Bilder zu starten, müssen die Bilder im Verzeichnis `HOME:` des aktiven Systems oder direkt im aktiven System abgelegt werden.

Es wird empfohlen, die Dateien im Verzeichnis `HOME:` abzulegen, so dass sie bei einem Backup and Restore gespeichert werden.

Ein **Neustart** muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird seine Größe geändert, so dass nur 28*28 Pixel angezeigt werden.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

Verwenden Sie die leere Zeichenfolge "" oder "stEmpty", wenn kein Icon angezeigt werden soll.

Komponente `text` des Typs `string`:

- Der Text der anzuzeigenden Menüzeile.
- Max. 75 Zeichen für jedes Listenmenüobjekt.

`[\Buttons]`

Datentyp: `buttondata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Gibt die anzuzeigenden Schaltflächen an. Es kann nur eine der vordefinierten Schaltflächenkombinationen des Typs buttondata verwendet werden. Siehe *Vordefinierte Daten auf Seite 1601*.

`[\BtnArray]`

Button Array

Datentyp: `string`

Eigene Definition der Schaltflächen, die in einem Datenfeld aus Zeichenfolgen gespeichert wird. Diese Funktion gibt den Datenfeldindex zurück, wenn die entsprechende Zeichenfolge ausgewählt ist.

Es kann jeweils nur der Parameter `\Buttons` oder `\BtnArray` verwendet werden. Wenn weder die `\Buttons` oder `\BtnArray` noch das Argument `\Buttons:=btnNone` verwendet werden, wird die Menüliste sofort entsprechend der Benutzerauswahl geändert.

Max. 5 Objekte, bestehend aus jeweils 42 Zeichen.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden.

Standardmäßig wird kein Icon angezeigt. Siehe *Vordefinierte Daten auf Seite 1601*.

`[\DefaultIndex]`

Datentyp: `num`

Standardmäßig entspricht die Benutzerauswahl im Listenmenü dem Index im im Parameter `ListItems` angegebenen Datenfeld.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn innerhalb dieser Zeitdauer keine Schaltfläche betätigt und keine Auswahl getroffen wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls keine Schaltfläche betätigt und keine Auswahl getroffen wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOBreak]()`

Digital Output Break

Datentyp: `signaldo`

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls keine Schaltfläche betätigt und keine Auswahl getroffen wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOPassive]`

Digital Output Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DOBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DOBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolBreak]`

Persistent Boolean Break

Datentyp: `bool`

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolPassive]`

Persistent Boolean Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `PersBoolBreak` verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn die persistente boolesche Variable `PersBoolBreak` auf

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignal verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Die Menüliste mit Icon, der Header, die Listenobjekte und das Standardobjekt werden entsprechend den programmierten Argumenten angezeigt. Die Programmabarbeitung wartet, bis der Bediener eine Auswahl getroffen hat oder die Menüliste durch einen Timeout oder eine Signalaktion unterbrochen wird. Das ausgewählte Listenobjekt und der Grund der Unterbrechung werden zurück in das Programm übertragen.

Die neue Menüliste auf Interrupt-Ebene übernimmt den Fokus von der Menüliste auf der Basisebene.

Vordefinierte Daten

```
!Icons:
CONST icondata iconNone := 0;
CONST icondata iconInfo := 1;
CONST icondata iconWarning := 2;
CONST icondata iconError := 3;
!Buttons:
CONST buttondata btnNone := -1;
CONST buttondata btnOK := 0;
CONST buttondata btnAbrtRtryIgn := 1;
CONST buttondata btnOKCancel := 2;
CONST buttondata btnRetryCancel := 3;
CONST buttondata btnYesNo := 4;
CONST buttondata btnYesNoCancel := 5;
!Results:
CONST btnres resUnkwn := 0;
CONST btnres resOK := 1;
CONST btnres resAbort := 2;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
CONST btnres resRetry := 3;
CONST btnres resIgnore := 4;
CONST btnres resCancel := 5;
CONST btnres resYes := 6;
CONST btnres resNo := 7;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion UIListView.

Beispiel 1:

```
CONST listitem list{2} := [ ["","Calibrate tool1"], ["","Calibrate
     tool2"] ];
VAR num list_item;
VAR errnum err_var;
...
list_item := UIListView
  ( \Header:="Select tool ?",
  list \Icon:=iconInfo
  \MaxTime:=60
  \DIBreak:=di5
  \BreakFlag:=err_var);
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
  CASE 0:
    ! Operator answer
    IF list_item =1 THEN
      ! Calibrate tool1
    ELSEIF list_item=2 THEN
      ! Calibrate tool2
    ENDIF
  DEFAULT:
    ! Not such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt und der Bediener kann ein Objekt in der Liste auswählen. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Im Programm ist die Ursache ersichtlich und es können entsprechende Maßnahmen eingeleitet werden.

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `UIListView` häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
UIListView '('
  [['\' Result ':=' <var or pers (INOUT) of btnres>]
  ['\' Header ':=' <expression (IN) of string>] ','']
  [ListItems '='] <array {*} (IN) of listitem>
  ['\' Buttons ':=' <expression (IN) of buttondata>]
  | ['\' BtnArray ':=' <array {*} (IN) of string>]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' DefaultIndex ':=' <expression (IN) of num>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\' PersBoolPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.214 UIListView - Anzeigen der Benutzerliste

RobotWare - OS

Fortsetzung

Information über	Siehe
Drucktastendaten	*buttondata – Drucktastendaten auf Seite 1642*
Ergebnisdaten für Drucktaste	*btnres - Ergebnisdaten für Drucktaste auf Seite 1639*
Datenstruktur des Listenobjekts	*listitem – Datenstruktur des Listenobjekts auf Seite 1722*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.215 UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ

Verwendung

`UIMessageBox` (*User Interaction Message Box*) wird verwendet, um mit dem Benutzer des Robotersystems über ein verfügbares Benutzergerät wie das FlexPendant zu kommunizieren. An den Bediener wird eine Meldung ausgegeben, der durch Drücken einer Taste antwortet. Die Benutzerauswahl wird dann zurück an das Programm übertragen.

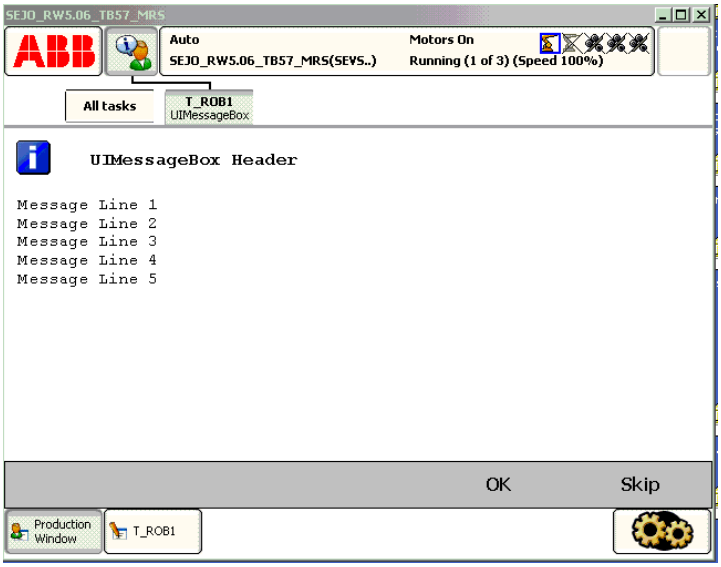
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UIMessageBox`.

Siehe auch *Weitere Beispiele auf Seite 1611*.

Beispiel 1:

```
VAR btnres answer;
CONST string my_message{5}:= ["Message Line 1","Message Line 2",
     "Message Line 3","Message Line 4","Message Line 5"];
CONST string my_buttons{2}:=["OK","Skip"];
...
answer:= UIMessageBox (
  \Header:="UIMessageBox Header"
  \MsgArray:=my_message
  \BtnArray:=my_buttons
  \Icon:=iconInfo);
IF answer = 1 THEN
  ! Operator selection OK
ELSEIF answer = 2 THEN
  ! Operator selection Skip
ELSE
  ! No such case defined
ENDIF
```



xx0500002409

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.215 UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ

RobotWare - OS

Fortsetzung

Das Meldungsfenster wird mit Icon, Header, Meldung und benutzerdefinierten Schaltflächen auf FlexPendant angezeigt. Die Programmabarbeitung wartet, bis OK oder „Weiter“ gedrückt wird. Mit anderen Worten, `answer` wird der Wert 1 (OK) oder 2 (Weiter) zugewiesen, je nachdem, welche Schaltfläche gedrückt wird (dem Datenfeldindex entsprechend).

Hinweis

`Message Line 1` bis `Message Line 5` werden auf separaten Zeilen 1 bis 5 angezeigt (der Schalter `\Wrap` wird nicht verwendet).

Rückgabewert

Datentyp: `btnres`

Der numerische Wert der Schaltfläche, die aus dem Meldungsfenster ausgewählt wurde.

Wenn das Argument `\Buttons` verwendet wird, werden die vordefinierten symbolischen Konstanten des Typs `btnres` zurückgegeben.

Wenn das Argument `\BtnArray` verwendet wird, wird der entsprechende Datenfeldindex zurückgegeben.

Brechen der Funktion über `\BreakFlag` oder bei `\Buttons:=btnNone`:

- Wenn der Parameter `\DefaultBtn` angegeben ist, wird dieser Index zurückgegeben.
- Wenn der Parameter `\DefaultBtn` nicht angegeben ist, wird `resUnkwn` gleich 0 zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine gebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UIMessageBox ( [\Header] [\Message] | [\MsgArray] [\Wrap][\Buttons]
     | [\BtnArray] [\DefaultBtn] [\Icon][\Image] [\MaxTime]
     [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive]
     [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
     [\UIActiveSignal])
```

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`[\Message]`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Max. 55 Zeichen.

`[\MsgArray]`

Message Array

Datentyp: `string`

Several text lines from an array to be written on the display.

Es kann jeweils nur der Parameter `\Message` oder `\MsgArray` verwendet werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der maximal verfügbare Platz beträgt 11 Zeilen mit jeweils 55 Zeichen.

[\Wrap]

Datentyp: switch

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument \MsgArray zu einer einzigen Zeichenfolge mit einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument \MsgArray in einer eigenen Zeile auf dem Display angezeigt.

[\Buttons]

Datentyp: buttondata

Defines the push buttons to be displayed. Only one of the predefined buttons combination of type buttondata can be used. Siehe *Vordefinierte Daten auf Seite 1610*.

Standardmäßig zeigt das System die Taste OK an.

[\BtnArray]

Button Array

Datentyp: string

Eigene Definition der Schaltflächen, die in einem Datenfeld aus Zeichenfolgen gespeichert wird. Diese Funktion gibt den Datenfeldindex zurück, wenn die entsprechende Zeichenfolge ausgewählt ist.

Es kann jeweils nur ein Parameter \Buttons oder \BtnArray gleichzeitig verwendet werden.

Max. 5 Objekte, bestehend aus jeweils 42 Zeichen.

[\DefaultBtn]

Default Button

Datentyp: btnres

Ermöglicht die Angabe eines Werts, der zurückgegeben werden soll, wenn das Meldungsfenster durch \MaxTime, \DIBreak oder \DOBreak unterbrochen wird. Es kann eine vordefinierte, symbolische Konstante des Typs btnres oder ein benutzerdefinierter Wert angegeben werden. Siehe *Vordefinierte Daten auf Seite 1610*.

[\Icon]

Datentyp: icondata

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs icondata verwendet werden. Siehe *Vordefinierte Daten auf Seite 1610*.

Standardmäßig wird kein Icon angezeigt.

[\Image]

Datentyp: string

Der Name des zu verwendenden Bildes. Um eigene Bilder zu starten, müssen die Bilder im Verzeichnis HOME: des aktiven Systems oder direkt im aktiven System abgelegt werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.215 UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ
RobotWare - OS
Fortsetzung

Es wird empfohlen, die Dateien im Verzeichnis HOME: abzulegen, so dass sie bei einem Backup and Restore gespeichert werden.

Ein **Neustart** muss durchgeführt werden. Anschließend lädt das FlexPendant die Bilder.

Voraussetzung für das System ist, dass die RobotWare-Option *FlexPendant Interface* verwendet wird.

Das angezeigte Bild kann über eine Breite von 185 Pixel und eine Höhe von 300 Pixel verfügen. Wenn das Bild größer ist, wird nur ein Bereich von 185 x 300 Pixel des Bildes ab der oberen linken Ecke des Bildes angezeigt.

Es kann kein genauer Wert für die zulässige Größe eines Bildes oder die Anzahl der Bilder angegeben werden, die auf das FlexPendant geladen werden können. Dies hängt von der Größer anderer Dateien ab, die auf das FlexPendant geladen wurden. Die Programmabarbeitung wird fortgesetzt, wenn ein Bild verwendet wird, das nicht auf das FlexPendant geladen wurde.

[\MaxTime]

Datentyp: num

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn innerhalb dieser Zeitdauer keine Taste gewählt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_MAXTIME kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

[\DIBreak]

Digital Input Break

Datentyp: signaldi

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DIPassive]

Digital Input Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DIBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DIBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DIBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOPassive]`

Digital Output Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DOBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DOBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DOBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolBreak]`

Persistent Boolean Break

Datentyp: `bool`

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\PersBoolPassive]`

Persistent Boolean Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `PersBoolBreak` verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn die persistente boolesche Variable `PersBoolBreak` auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten `ERR_TP_PERSBOOLBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\BreakFlag]`

Datentyp: `errnum`

Eine Variable, die den Fehlercode enthält, wenn `MaxTime`, `DIBreak`, `DOBreak` oder `PersBoolBreak` verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten `ERR_TP_MAXTIME`, `ERR_TP_DIBREAK`, `ERR_TP_DOBREAK` und `ERR_TP_PERSBOOLBREAK` können verwendet werden, um die Ursache zu bestimmen.

`[\UIActiveSignal]`

Datentyp: `signaldo`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.215 UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ
RobotWare - OS
Fortsetzung

Das digitale Ausgangssignal, das im optionalen Argument `UIActiveSignal` verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Das Meldungsfenster mit dem Icon, dem Header, den Meldungszeilen, Bild und Tasten wird gemäß den programmierten Argumenten angezeigt. Die Programmabarbeitung wartet, bis der Benutzer eine Taste betätigt oder das Meldungsfenster durch einen Timeout oder ein Signal unterbrochen wird. Die Auswahl des Benutzers und der Grund der Unterbrechung werden in das Programm zurückübertragen.

Ein neues Meldungsfenster auf Interrupt-Ebene übernimmt den Fokus vom Meldungsfenster auf der Basisebene.

Vordefinierte Daten

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
!Buttons:
  CONST buttondata btnNone := -1;
  CONST buttondata btnOK := 0;
  CONST buttondata btnAbrtRtryIgn := 1;
  CONST buttondata btnOKCancel := 2;
  CONST buttondata btnRetryCancel := 3;
  CONST buttondata btnYesNo := 4;
  CONST buttondata btnYesNoCancel := 5;
!Results:
  CONST btnres resUnkwn := 0;
  CONST btnres resOK := 1;
  CONST btnres resAbort := 2;
  CONST btnres resRetry := 3;
  CONST btnres resIgnore := 4;
  CONST btnres resCancel := 5;
  CONST btnres resYes := 6;
  CONST btnres resNo := 7;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion UIMessageBox.

Beispiel 1:

```
VAR errnum err_var;
VAR btnres answer;
...
answer := UIMessageBox (\Header:= "Cycle step 3"
  \Message:="Continue with the calibration ?" \Buttons:=btnOKCancel
  \DefaultBtn:=resCancel \Icon:=iconInfo \MaxTime:=60 \DIBreak:=di5
  \BreakFlag:=err_var);
IF answer = resOK THEN
  ! OK from the operator
ELSE
  ! Cancel from the operator or operation break
  TEST err_var
    CASE ERR_TP_MAXTIME:
      ! Time out
    CASE ERR_TP_DIBREAK:
      ! Input signal break
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.215 UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ

RobotWare - OS

Fortsetzung

```
    DEFAULT:
      ! Not such case defined
  ENDTEST
ENDIF
```

Das Meldungsfenster wird angezeigt und der Bediener kann OK oder Abbrechen auswählen. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Die Gründe werden im Programm angegeben.

Einschränkungen

Setzen Sie für den Timeout-Parameter `\MaxTime` keinen zu niedrigen Wert ein, wenn `UIMessageBox` häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
UIMessageBox '('
  ['\' Header ':=' <expression (IN) of string>] ',']
  ['\' Message ':=' <expression (IN) of string>]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Buttons ':=' <expression (IN) of buttondata>]
  | ['\' BtnArray ':=' <array {*} (IN) of string>]
  ['\' DefaultBtn ':=' <expression (IN) of btnres>]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' Image ':=' <expression (IN) of string>]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\' PersBoolPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `btnres`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Drucktastendaten	*buttondata – Drucktastendaten auf Seite 1642*
Ergebnisdaten für Drucktaste	*btnres - Ergebnisdaten für Drucktaste auf Seite 1639*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
FlexPendant interface	*Produktspezifikation - Steuerungssoftware IRC5*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.216 UINumEntry - Eintrag eines numerischen Werts durch den Benutzer

Verwendung

`UINumEntry` (*User Interaction Number Entry*) ermöglicht dem Bediener die Eingabe eines numerischen Wertes auf dem verfügbaren Benutzergerät, z. B. dem FlexPendant. An den Bediener wird eine Meldung ausgegeben, und dieser antwortet mit einem numerischen Wert. Anschließend wird der numerische Wert überprüft, bestätigt und zurück an das Programm übertragen.

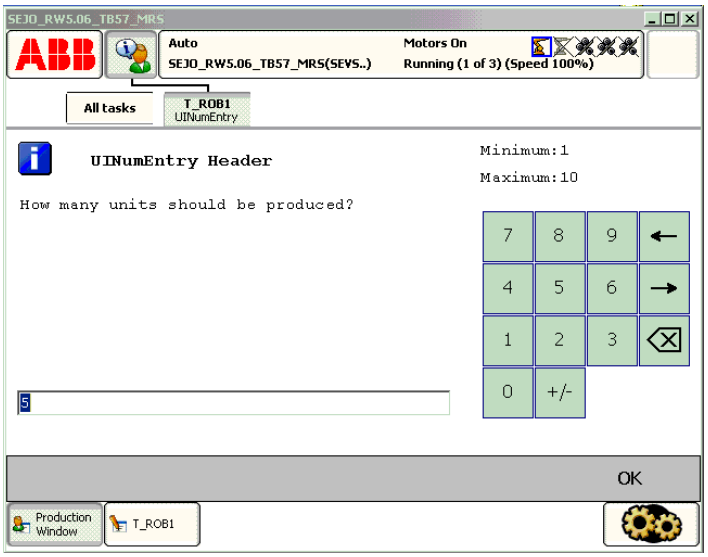
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UINumEntry`.

Siehe auch *Weitere Beispiele auf Seite 1619*.

Beispiel 1:

```
VAR num answer;
...
answer := UINumEntry(
  \Header:="UINumEntry Header"
  \Message:="How many units should be produced?"
  \Icon:=iconInfo
  \InitValue:=5
  \MinValue:=1
  \MaxValue:=10
  \AsInteger);
FOR i FROM 1 TO answer DO
  produce_part;
ENDFOR
```



xx0500002412

Das numerische Meldungsfenster wird mit Icon Anfangs-, Maximal- und Minimalwert auf FlexPendant angezeigt. Im Meldungsfenster wird überprüft, ob der Bediener eine Ganzzahl im Wertebereich ausgewählt hat. Die Programmabarbeitung wartet, bis OK gedrückt wurde und dann der gewählte numerische Wert zurückgegeben

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wird. Die Routine `produce_part` wird dann entsprechend der am FlexPendant eingegebenen Zahl wiederholt.

Rückgabewert

Datentyp: `num`

Diese Funktion gibt den numerischen Eingangswert zurück.

Brechen der Funktion über `\BreakFlag`:

- Wenn der Parameter `\InitValue` angegeben ist, wird dieser Wert zurückgegeben.
- Wenn der Parameter `\InitValue` nicht angegeben ist, wird der Wert 0 zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine gebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UINumEntry ( [\Header] [\Message] | [\MsgArray]
     [\Wrap][\Icon][\InitValue] [\MinValue] [\MaxValue]
     [\AsInteger][\MaxTime] [\DIBreak] [\DIPassive] [\DOBreak]
     [\DOPassive] [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
     [\UIActiveSignal])
```

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`[\Message]`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Max. 40 Zeichen.

`[\MsgArray]`

Message Array

Datentyp: `string`

Several text lines from an array to be written on the display.

Es kann jeweils nur der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der maximal verfügbare Platz beträgt 9 Zeilen mit jeweils 40 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile am Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.216 UINumEntry - Eintrag eines numerischen Werts durch den Benutzer
RobotWare - OS
Fortsetzung

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1618*.

Standardmäßig wird kein Icon angezeigt.

`[\InitValue]`

Datentyp: `num`

Der Anfangswert, der im Eingabefenster angezeigt wird.

`[\MinValue]`

Datentyp: `num`

Der Mindestwert für den Rückgabewert.

`[\MaxValue]`

Datentyp: `num`

Der Maximalwert für den Rückgabewert.

`[\AsInteger]`

Datentyp: `switch`

Entfernt den Dezimalpunkt vom Nummernblock, um sicherzustellen, dass der Rückgabewert eine Ganzzahl ist.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn die OK-Taste innerhalb dieser Zeitdauer nicht betätigt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

[\DOBreak]

Digital Output Break

Datentyp: signaldo

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DOBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolBreak]

Persistent Boolean Break

Datentyp: bool

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolPassive]

Persistent Boolean Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument PersBoolBreak verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn die persistente boolesche Variable PersBoolBreak auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.216 UINumEntry - Eintrag eines numerischen Werts durch den Benutzer
RobotWare - OS
Fortsetzung

ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignal verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Das numerische Meldungsfenster mit Nummernblock, Icon, Header, Meldungszeilen, Anfangs-, Maximal- und Minimalwert wird den programmierten Argumenten entsprechend angezeigt. Die Programmabarbeitung wartet, bis der Benutzer einen zulässigen numerischen Wert eingibt und dann die OK-Taste betätigt oder das Meldungsfenster durch ein Timeout oder eine Signalaktion unterbrochen wird. Der numerische Wert und der Grund der Unterbrechung werden in das Programm zurückübertragen.

Neue Meldungsfenster auf Interrupt-Ebene werden von Meldungsfenstern auf Basisebene beachtet.

Vordefinierte Daten

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable und nicht mit einem E/A-Signal verbunden, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.
ERR_UI_INITVALUE	Der Anfangswert (Parameter \InitValue) liegt nicht zwischen dem Minimal- und Maximalwert (Parameter \MinValue und \MaxValue).
ERR_UI_MAXMIN	Der Mindestwert (Parameter \MinValue) ist größer als der Maximalwert (Parameter \MaxValue).
ERR_UI_NOTINT	Der Anfangswert (Parameter \InitValue) ist keine Ganzzahl, wie im Parameter \AsInteger festgelegt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion UINumEntry.

Beispiel 1:

```
VAR errnum err_var;
VAR num answer;
VAR num distance;
...
answer := UINumEntry (\Header:= "BWD move on path"
  \Message:="Enter the path overlap ?" \Icon:=iconInfo
  \InitValue:=5 \MinValue:=0 \MaxValue:=10
  \MaxTime:=60 \DIBreak:=di5 \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer distance := 5;
  CASE 0
    ! Operator answer
    distance := answer;
  DEFAULT:
    ! Not such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt und der Bediener kann einen numerischen Wert eingeben und OK drücken. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Im Programm ist die Ursache ersichtlich und es können entsprechende Maßnahmen eingeleitet werden.

Einschränkungen

Setzen Sie für den Timeout-Parameter \MaxTime keinen zu niedrigen Wert ein, wenn UINumEntry häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.216 UINumEntry - Eintrag eines numerischen Werts durch den Benutzer

RobotWare - OS

Fortsetzung

Syntax

```
UINumEntry '('
  ['\' Header ':=' <expression (IN) of string>]
  [Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':=' <array {*} (IN) of string>]
  ['\' Wrap]
  ['\' Icon ':=' <expression (IN) of icondata>]
  ['\' InitValue ':=' <expression (IN) of dnum>]
  ['\' MinValue ':=' <expression (IN) of dnum>]
  ['\' MaxValue ':=' <expression (IN) of dnum>]
  ['\' AsInteger]
  ['\' MaxTime ':=' <expression (IN) of num>]
  ['\' DIBreak ':=' <variable (VAR) of signaldi>]
  ['\' DIPassive]
  ['\' DOBreak ':=' <variable (VAR) of signaldo>]
  ['\' DOPassive]
  ['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
  ['\' PersBoolPassive]
  ['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
  ['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahlensynchronisierung“	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.217 UINumTune - Benutzer-Zahlensynchronisierung

Verwendung

`UINumTune` (*User Interaction Number Tune*) ermöglicht dem Bediener das Abstimmen eines numerischen Wertes auf dem verfügbaren Benutzergerät, z.B. dem FlexPendant. An den Bediener wird eine Meldung ausgegeben, und dieser stimmt einen numerischen Wert ab. Anschließend wird der abgestimmte numerische Wert überprüft, bestätigt und zurück an das Programm übertragen.

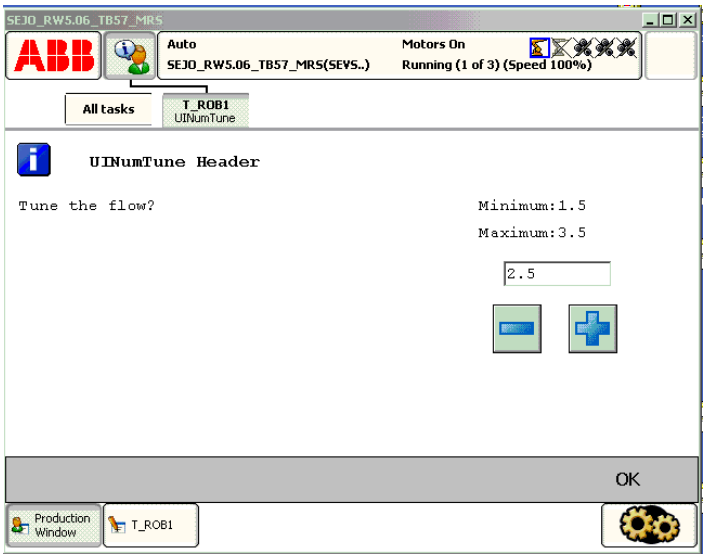
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `UINumTune`.

Siehe auch *Weitere Beispiele auf Seite 1626*.

Beispiel 1:

```
VAR num flow;
...
flow := UINumTune(
  \Header:="UINumTune Header"
  \Message:="Tune the flow?"
  \Icon:=iconInfo,
  2.5,
  0.1
  \MinValue:=1.5
  \MaxValue:=3.5);
```



xx0500002414

Das Abstimmungs-Meldungsfenster wird mit Icon Header, Meldung, Anfangs-, Inkremental-, Maximal- und Minimalwert auf FlexPendant angezeigt. Im Meldungsfenster wird überprüft, ob der Bediener den Flow-Wert mit der Schrittgröße 0.1 vom Anfangswert 2.5 im Wertebereich 1.5-3.5abgestimmt hat. Die Programmabarbeitung wartet, bis auf OK getippt wird und dann der gewählte numerische Wert zurückgegeben und in der Flow-Variablen gespeichert wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.217 UINumTune - Benutzer-Zahlensynchronisierung

RobotWare - OS

Fortsetzung

Rückgabewert

Datentyp: `num`

Diese Funktion gibt den abgestimmten numerischen Wert zurück.

Wenn die Funktion über ein `\BreakFlag` gebrochen wird, wird der angegebene `InitValue` zurückgegeben.

Wenn die Funktion über die `ERROR`-Behandlungsroutine gebrochen wird, wird kein Wert zurückgegeben.

Argumente

```
UINumTune ( [\Header] [\Message] | [\MsgArray] [\Wrap] [\Icon]
     InitValue Increment [\MinValue] [\MaxValue] [\MaxTime]
     [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive]
     [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
     [\UIActiveSignal])
```

`[\Header]`

Datentyp: `string`

Die Überschrift des Meldungsfensters (max. 40 Zeichen).

`[\Message]`

Datentyp: `string`

Eine Textzeile, die am Display angezeigt wird. Max. 40 Zeichen.

`[\MsgArray]`

Message Array

Datentyp: `string`

Mehrere Textzeilen von einem Datenfeld, die am Display angezeigt werden.

Es kann jeweils nur der Parameter `\Message` oder `\MsgArray` verwendet werden.

Der maximal verfügbare Platz beträgt 11 Zeilen mit jeweils 40 Zeichen.

`[\Wrap]`

Datentyp: `switch`

Wenn dieses Argument ausgewählt ist, werden alle angegebenen Zeichenfolgen im Argument `\MsgArray` zu einer einzigen Zeichenfolge mit einem einzelnen Leerzeichen zwischen den einzelnen Zeichenfolgen verkettet und auf eine möglichst geringe Anzahl an Zeilen verteilt.

Standardmäßig wird jede Zeichenfolge im Argument `\MsgArray` in einer eigenen Zeile am Display angezeigt.

`[\Icon]`

Datentyp: `icondata`

Definiert das anzuzeigende Icon. Es kann nur eines der vordefinierten Icons des Typs `icondata` verwendet werden. Siehe *Vordefinierte Daten auf Seite 1625*.

Standardmäßig wird kein Icon angezeigt.

`InitValue`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Anfangswert, der im Eingabefenster angezeigt wird.

`Increment`

Datentyp: `num`

Dieser Parameter gibt an, um wie viel sich der Wert ändern soll, wenn die Plus- oder Minus-Schaltfläche betätigt wird.

`[\MinValue]`

Datentyp: `num`

Der Mindestwert für den Rückgabewert.

`[\MaxValue]`

Datentyp: `num`

Der Maximalwert für den Rückgabewert.

`[\MaxTime]`

Datentyp: `num`

Die maximale Zeitdauer in Sekunden, für die die Programmabarbeitung wartet. Wenn die OK-Taste innerhalb dieser Zeitdauer nicht betätigt wird, fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_MAXTIME` kann überprüft werden, ob die maximale Zeitdauer verstrichen ist.

`[\DIBreak]`

Digital Input Break

Datentyp: `signaldi`

Das digitale Eingangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn, `BreakFlag` wird verwendet (siehe unten). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DIPassive]`

Digital Input Passive

Datentyp: `switch`

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument `DIBreak` verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein `BreakFlag` verwendet wird), wenn das Signal `DIBreak` auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten `ERR_TP_DIBREAK` kann überprüft werden, ob dies eingetreten ist.

`[\DOBreak]`

Digital Output Break

Datentyp: `signaldo`

Das digitale Ausgangssignal, das den Bedienerdialog unterbrechen kann. Falls die OK-Taste nicht betätigt wird, wenn das Signal auf 1 gesetzt wird (oder bereits auf 1 gesetzt ist), fährt das Programm in der Fehlerbehandlung fort, es sei denn,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.217 UINumTune - Benutzer-Zahlensynchronisierung
RobotWare - OS
Fortsetzung

BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\DOPassive]

Digital Output Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument DOBreak verwendet wird. Anstatt zu reagieren, wenn das Signal auf 1 gesetzt wird (oder bereits 1 ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn das Signal DOBreak auf 0 gesetzt wird (oder bereits 0 ist). Mithilfe der Konstanten ERR_TP_DOBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolBreak]

Persistent Boolean Break

Datentyp: bool

Die persistente boolesche Variable, die den Bedienerdialog unterbrechen kann. Falls keine Taste ausgewählt wird, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE steht), fährt das Programm in der Fehlerbehandlung fort, es sei denn, BreakFlag wird verwendet (siehe unten). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\PersBoolPassive]

Persistent Boolean Passive

Datentyp: switch

Der Schalter hat Vorrang vor dem Standardverhalten, wenn das optionale Argument PersBoolBreak verwendet wird. Anstatt zu reagieren, wenn die persistente boolesche Variable auf TRUE gesetzt wird (oder bereits auf TRUE gesetzt ist), sollte die Instruktion mit der Fehlerbehandlung fortfahren (wenn kein BreakFlag verwendet wird), wenn die persistente boolesche Variable PersBoolBreak auf FALSE gesetzt wird (oder bereits FALSE ist). Mithilfe der Konstanten ERR_TP_PERSBOOLBREAK kann überprüft werden, ob dies eingetreten ist.

[\BreakFlag]

Datentyp: errnum

Eine Variable, die den Fehlercode enthält, wenn MaxTime, DIBreak, DOBreak oder PersBoolBreak verwendet wird. Wenn diese optionale Variable weggelassen wird, wird die Fehlerbehandlung abgearbeitet. Die Konstanten ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK und ERR_TP_PERSBOOLBREAK können verwendet werden, um die Ursache zu bestimmen.

[\UIActiveSignal]

Datentyp: signaldo

Das digitale Ausgangssignal, das im optionalen Argument UIActiveSignal verwendet wird, wird auf 1 gesetzt, wenn die Meldungsbox am FlexPendant aktiviert

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wurde. Wenn die Benutzerauswahl erfolgt ist und die Ausführung fortgesetzt wird, wird das Signal wieder auf 0 gesetzt.

Es ist keine Überwachung von Stopp oder Neustart vorhanden. Das Signal wird auf 0 gesetzt, wenn die Funktion bereit ist oder wenn der PP bewegt wird.

Programmabarbeitung

Das numerische Abstimmungs-Meldungsfenster mit den Abstimmungsschaltflächen +/-, Icon, Header, Meldungszeilen, Anfangs-, Inkremental-, Maximal- und Minimalwert wird den programmierten Argumenten entsprechend angezeigt. Die Programmabarbeitung wartet, bis der Benutzer den numerischen Wert abgestimmt hat und dann die OK-Taste betätigt oder das Meldungsfenster durch ein Timeout oder eine Signalaktion unterbrochen wird. Der numerische Wert und der Grund der Unterbrechung werden in das Programm zurückübertragen.

Neue Meldungsfenster auf Interrupt-Ebene werden von Meldungsfenstern auf Basisebene beachtet.

Vordefinierte Daten

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable ERRNO wird gesetzt auf:

Name	Fehlerursache
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_TP_NO_CLIENT	Kein Client, z. B. ein FlexPendant, um die Instruktion zu behandeln.
ERR_UI_INITVALUE	Der Anfangswert (Parameter \InitValue) liegt nicht zwischen dem Minimal- und Maximalwert (Parameter \MinValue und \MaxValue).
ERR_UI_MAXMIN	Der Mindestwert (Parameter \MinValue) ist größer als der Maximalwert (Parameter \MaxValue).

Wenn der Parameter \BreakFlag nicht verwendet wird, können folgende Situationen über die Fehlerbehandlungsroutine behoben werden:

- Wenn vor einer Bedienereingabe ein Timeout (Parameter \MaxTime) eintritt, wird die Systemvariable ERRNO auf ERR_TP_MAXTIME gesetzt und die Abarbeitung fährt in der Fehlerbehandlung fort.
- Wenn vor einer Bedienereingabe ein digitaler Eingang gesetzt wird (Parameter \DIBreak), wird die Systemvariable ERRNO auf ERR_TP_DIBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Wenn vor einer Bedienereingabe ein digitaler Ausgang gesetzt wird (Parameter \DOBreak), wird die Systemvariable ERRNO auf ERR_TP_DOBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.
- Wenn vor einer Bedienereingabe eine persistente boolesche Variable (Parameter \PersBoolBreak) gesetzt wird, dann wird die Systemvariable ERRNO auf ERR_TP_PERSBOOLBREAK gesetzt und die Abarbeitung fährt in der Fehlerbehandlungsroutine fort.

Weitere Beispiele

Das folgende Beispiel veranschaulicht die Funktion UINumTune.

Beispiel 1:

```
VAR errnum err_var;
VAR num tune_answer;
VAR num distance;
...
tune_answer := UINumTune (\Header:=" BWD move on path"
     \Message:="Enter the path overlap ?" \Icon:=iconInfo, 5, 1
     \MinValue:=0 \MaxValue:=10 \MaxTime:=60 \DIBreak:=di5
     \BreakFlag:=err_var);
TEST err_var
  CASE ERR_TP_MAXTIME:
  CASE ERR_TP_DIBREAK:
    ! No operator answer
    distance := 5;
  CASE 0:
    ! Operator answer
    distance := tune_answer;
  DEFAULT:
    ! No such case defined
ENDTEST
```

Das Meldungsfenster wird angezeigt und der Bediener kann einen numerischen Wert abstimmen und OK drücken. Das Meldungsfenster kann auch durch ein Timeout oder ein digitales Eingangssignal unterbrochen werden. Im Programm ist die Ursache ersichtlich und es können entsprechende Maßnahmen eingeleitet werden.

Einschränkungen

Setzen Sie für den Timeout-Parameter \MaxTime keinen zu niedrigen Wert ein, wenn UINumTune häufig , z. B. in einer Schleife, ausgeführt wird. Dies kann zu nicht vorhersehbarem Verhalten der Systemleistung, z. B. zu einer langsamen Reaktion des FlexPendant, führen.

Syntax

```
UINumTune '('
  ['\' Header ':=' <expression (IN) of string>]
  [Message ':=' <expression (IN) of string> ]
  | ['\' MsgArray ':='<array {*} (IN) of string>]
  ['\' Wrap]
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
['\' Icon ':=' <expression (IN) of icondata>]
[InitValue ':=' <expression (IN) of num>]
[Increment ':=' <expression (IN) of num>]
['\' MinValue ':=' <expression (IN) of num>]
['\' MaxValue ':=' <expression (IN) of num>]
['\' MaxTime ':=' <expression (IN) of num>]
['\' DIBreak ':=' <variable (VAR) of signaldi>]
['\' DIPassive]
['\' DOBreak ':=' <variable (VAR) of signaldo>]
['\' DOPassive]
['\' PersBoolBreak ':=' <persistent (PERS) of bool>]
['\' PersBoolPassive]
['\' BreakFlag ':=' <var or pers (INOUT) of errnum>]
['\' UIActiveSignal ':=' <variable (VAR) of signaldo>] ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Daten für Icon-Anzeige	*icondata – Daten für Icon-Anzeige auf Seite 1712*
Benutzerinteraktions-Meldungsfenster, grundlegender Typ	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster, erweiterter Typ	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion „Zahleneingabe“	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion „Alpha-Eingabe“	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion „Listenansicht“	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
System verbunden mit FlexPendant usw.	*UIClientExist - Vorhandensein eines Benutzer-Clients auf Seite 1578*
Löschen des Inhalts im Bedienerfenster	*TPErase - Löschen des Texts am FlexPendant auf Seite 886*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.218 ValidIO - Gültiges E/A-Signal, auf das zugegriffen werden soll

Verwendung

Mit `ValidIO` wird überprüft, ob gegenwärtig auf das angegebene E/A-Signal ohne Fehler zugegriffen werden kann.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `ValidIO`.

Beispiel 1:

```
IF ValidIO(mydosignal) SetDO mydosignal, 1;
```

Setzen Sie das digitale Ausgangssignal `mydosignal` auf`1`, wenn das zugehörige E/A-Gerät in Betrieb ist.

Rückgabewert

Datentyp: `bool`

Gibt `TRUE` zurück, wenn das E/A-Signal gültig ist und das E/A-Gerät für das Signal in Betrieb ist.

Gibt `FALSE` zurück, wenn die E/A-Einheit nicht in Betrieb ist oder wenn keine `AliasIO`-Instruktion abgearbeitet wurde, um eine Signalvariable zu verbinden, die im RAPID-Programm für ein Signal angegeben wurde, das in der E/A-Konfiguration definiert wurde.

Argumente

```
ValidIO (Signal)
```

`Signal`

Datentyp: `signalxx`

Name des E/A-Signals. Muss vom Datentyp `signaldo, signaldi, signalgo, signalgi, signalao` oder `signalai` sein.

Programmabarbeitung

Abarbeitungsverhalten:

- Überprüfen, ob gültiges E/A-Signal
- Überprüfen, ob das E/A-Gerät für das Signal in Betrieb ist.

Es werden keine Fehlermeldungen generiert.

Syntax

```
ValidIO '('
  [Signal ':='] <variable (VAR) of anytype> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `bool`.

Weitere Informationen

Information über	Siehe
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*
Definieren eines E/A-Signals mit Aliasname	*AliasIO - Definieren eines E/A-Signals mit Aliasname auf Seite 37*
Leseattribut eines Systemparameters	*ReadCfgData - Lesen von Attributen eines Systemparameters auf Seite 586*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.219 ValToStr - Konvertieren eines Werts in eine Zeichenfolge

Verwendung

`ValToStr` (*Value To String*) wird verwendet, um einen Wert eines beliebigen Datentyps in eine Zeichenfolge zu konvertieren.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion `ValToStr`.

Beispiel 1:

```
VAR string str;
VAR pos p := [100,200,300];
str := ValToStr(p);
```

Die Variable `str` erhält den Wert `"[100,200,300]"`.

Beispiel 2:

```
str := ValToStr(TRUE);
```

Die Variable `str` erhält den Wert `TRUE`.

Beispiel 3

```
str := ValToStr(1.234567890123456789);
```

Die Variable `str` erhält den Wert `"1.23456789012346"`.

Beispiel 4:

```
VAR num numtype:=1.234567890123456789;

str := ValToStr(numtype);
```

Die Variable `str` erhält den Wert `"1.23457"`.

Beispiel 5:

```
VAR dnum dnumtype:=1.234567890123456789;

str := ValToStr(dnumtype);
```

Die Variable `str` erhält den Wert `"1.23456789012346"`.

Rückgabewert

Datentyp: `string`

Der Wert wird in eine Zeichenfolge mit `RAPID`-Standardformat konvertiert. Dies bedeutet im Prinzip 6 signifikante Stellen. Literalwerte werden als `dnum` (siehe Beispiel 3) und `dnum`-Variablen interpretiert, (siehe Beispiel 5), obwohl diese 15 signifikante Stellen aufweisen.

Ein Laufzeitfehler tritt auf, wenn die resultierende Zeichenfolge zu lang ist.

Argumente

```
ValToStr ( Val )
```

`Val`

Value

Datentyp: `anytype`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Ein Wert eines beliebigen Datentyps. Es kann ein beliebiger Typ von value-Daten verwendet werden, z. B. Einzeldaten, Datensätze, Datensatzkomponenten, Datenfelder oder Datenfeldkomponenten.

Syntax

```
ValToStr '('
  [ Val ':=' ] <expression (IN) of anytype> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `string`.

Weitere Informationen

Information über	Siehe
String-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Definition einer Zeichenfolge	*string – Zeichenfolgen auf Seite 1806*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.220 Vectmagn - Berechnen der Länge eines pos-Vektors

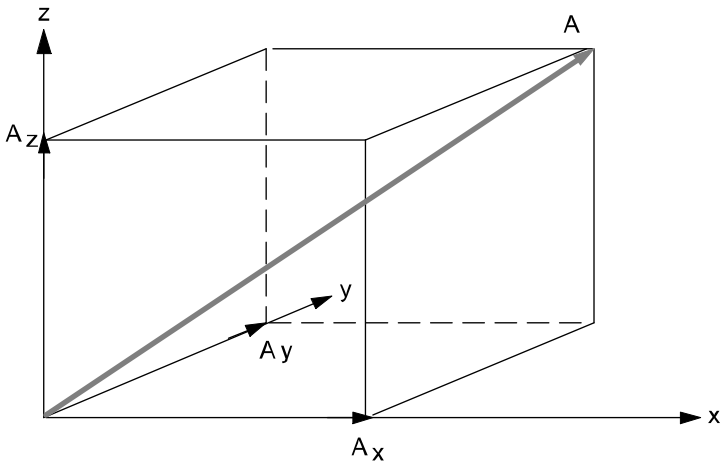
Verwendung

`VectMagn` (*Vector Magnitude*) wird verwendet, um die Länge eines `pos` -Vektors zu berechnen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die Funktion `VectMagn`.

Beispiel 1:



xx0500002446

Ein Vektor **A** kann als die Summe seiner Komponenten in drei orthogonalen Richtungen geschrieben werden:

$A=A_x x+A_y y+A_y y$

Die Länge von **A** ist:

$$|A| = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

Der Vektor wird durch den Datentyp `pos` und die Länge durch den Datentyp `num` beschrieben:

```
VAR num magnitude;
VAR pos vector;
...
vector := [1,1,1];
magnitude := VectMagn(vector);
```

Rückgabewert

Datentyp: `num`

Die Länge des Vektors (Datentyp `pos`).

Argumente

```
VectMagn (Vector)
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`Vector`

Datentyp: `pos`

Der Vektor mit dem Datentyp `pos`.

Syntax

```
VectMagn '('
  [Vector ':='] <expression (IN) of pos> ')'
```

Eine Funktion mit einem Rückgabewert des Typs `num`.

Weitere Informationen

Information über	Siehe
Mathematische Instruktionen und Funktionen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2.221 XOR – Bewertet einen logischen Wert

Verwendung

XOR(*Exclusive Or*) ist ein bedingter Ausdruck, der verwendet wird, um einen logischen Wert (true/false) zu errechnen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen die Funktion XOR.

Beispiel 1:

```
VAR bool a;
VAR bool b;
VAR bool c;
c := a XOR b;
```

Der Rückgabewert c istTRUE wenn nur ein einziger von a oder b TRUE ist. Ansonsten ist der Rückgabewert FALSE.

Beispiel 2:

```
VAR num a;
VAR num b;
VAR bool c;
...
c := a>5 XOR b=3;
```

Der Rückgabewert c ist TRUE, wenn nur eine einzige Bedingung TRUE ist. Entweder a ist größer als 5, oder b ist gleich 3. Ansonsten ist der Rückgabewert FALSE.

Rückgabewert

Datentyp: bool

Der Rückgabewert ist TRUE, wenn nur eine einzige der Bedingungen richtig ist. Ansonsten ist der Rückgabewert FALSE.

Syntax

```
<expression of bool> XOR <expression of bool>
```

Eine Funktion mit einem Rückgabewert des Typs bool.

Weitere Informationen

Information über	Siehe
AND	*AND – Bewertet einen logischen Wert auf Seite 1175*
OR	*OR – Bewertet einen logischen Wert auf Seite 1415*
NOT	*NOT – Kehrt einen logischen Wert um auf Seite 1406*
Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3 Datentypen

3.1 aiotrigg - Bedingung für analogen E/A-Trigger

Verwendung

`aiotrigg` (*Analog I/O Trigger*) definiert die Bedingung zur Erzeugung eines Interrupts für ein analoges Eingangs- oder Ausgangssignal.

Beschreibung

Daten des Typs `aiotrigg` definieren, wie anhand eines oberen und unteren Schwellwerts bestimmt wird, ob der logische Wert eines analogen Signals die Bedingung zur Erzeugung eines Interrupts erfüllt.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `aiotrigg`:

Beispiel 1:

```
VAR intnum sig1int;
PROC main()
  CONNECT sig1int WITH iroutine1;
  ISignalAI \Single, ai1, AIO_BETWEEN, 1.5, 0.5, 0, sig1int;
```

Fordert einen Interrupt an, wenn der logische Wert des analogen Eingangssignals `ai1` erstmals zwischen `0.5` und `1.5` liegt. Dann erfolgt ein Aufruf der Interrupt-Routine `iroutine1` .

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `aiotrigg` sind vordefiniert und ermöglichen die Angabe einer Bedingung für die Instruktionen `ISignalAI` und `ISignalAO`.

Wert	Symbolische Konstante	Kommentar
1	`AIO_ABOVE_HIGH`	Signal erzeugt Interrupts, wenn angegebener oberer Wert überschritten wird.
2	`AIO_BELOW_HIGH`	Signal erzeugt Interrupts, wenn angegebener oberer Wert unterschritten wird.
3	`AIO_ABOVE_LOW`	Signal erzeugt Interrupts, wenn angegebener unterer Wert überschritten wird.
4	`AIO_BELOW_LOW`	Signal erzeugt Interrupts, wenn angegebener unterer Wert unterschritten wird.
5	`AIO_BETWEEN`	Signal erzeugt Interrupts, wenn es zwischen dem angegebenen unteren und oberen Wert liegt.
6	`AIO_OUTSIDE`	Signal erzeugt Interrupts, wenn angegebener unterer Wert unterschritten oder oberer Wert überschritten wird.
7	`AIO_ALWAYS`	Signal erzeugt immer Interrupts.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.1 aiotrigg - Bedingung für analogen E/A-Trigger

RobotWare - OS

Fortsetzung

Eigenschaften

`aiotrigg` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Interrupt von analogem Eingangssignal	*ISignalAI - Interrupts von analogem Eingangssignal auf Seite 307*
Interrupt von analogem Ausgangssignal	*ISignalAO - Interrupts von analogem Ausgangssignal auf Seite 316*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.2 ALIAS – Zuweisen eines Alias-Datentyps

Verwendung

`ALIAS` wird verwendet, um einen Datentyp so zu definieren, dass er einem anderen Datentyp gleich ist. Alias-Typen bieten eine Möglichkeit, um Objekte zu klassifizieren. Das System verwendet die Alias-Klassifizierung eventuell zum Nachschlagen und Präsentieren von typbezogenen Objekten. Ein Alias-Typ wird von einer Alias-Definition eingeführt.

Die integrierten Alias-Typen sind `errnum` und `intnum`, beides Alias für `num`.

`errnum`-Typ

Der `errnum`-Typ ist ein Alias für `num` und wird zur Darstellung von Fehlerzahlen verwendet.

`intnum`-Typ

Der `intnum`-Typ ist ein Alias für `num` und wird zur Darstellung von Interrupt-Zahlen verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht die `ALIAS`-Definition:

Beispiel 1:

```
ALIAS num level;
CONST level low := 2.5;
CONST level high := 4.0;
```

Ein Alias-Typ `level` ist definiert (Alias für `num`).

Einschränkungen

Um von RAPID erkannt zu werden, müssen alle Alias-Definitionen ganz oben im Programm oder im Systemmodul vor allen anderen Datendeklarationen deklariert werden. Der einzige Datentyp, der vor dem Alias deklariert werden darf, ist `RECORD`.

Es darf kein Alias-Typ auf einem anderen Alias-Typ definiert werden.

Syntax

```
ALIAS <type name> <identifier> ';'
```

Alias-Definition.

Weitere Informationen

Information über	Siehe
errnum - Fehlernummer	*errnum – Fehlernummer auf Seite 1694*
intnum - Interrupt-Identität	*intnum - Interrupt-Identität auf Seite 1716*
Lexikalische Elemente	*Technical reference manual - RAPID kernel*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.3 bool – Logische Werte

Verwendung

`bool` wird für logische Werte (TRUE/FALSE) verwendet.

Beschreibung

Der Wert von Daten des Typs `bool` kann `TRUE` oder `FALSE` sein.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `bool` :

Beispiel 1:

```
flag1 := TRUE;
```

Flag erhält den Wert TRUE.

Beispiel 2:

```
VAR bool highvalue;
VAR num reg1;
...
highvalue := reg1 > 100;
```

`highvalue` erhält den Wert `TRUE`, wenn `reg1` größer als `100` ist; anderenfalls wird `FALSE` zugewiesen.

Beispiel 3

```
IF highvalue Set do1;
```

Das Signal `do1` wird gesetzt, wenn `highvalue` den Wert `TRUE` hat.

Beispiel 4:

```
highvalue := reg1 > 100;
mediumvalue := reg1 > 20 AND NOT highvalue;
```

`mediumvalue` erhält den Wert `TRUE`, wenn `reg1` zwischen `20` und `100` liegt.

Weitere Informationen

Information über	Siehe
Logische Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Expressions*
Operationen mit logischen Werten	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Expressions*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.4 btnres - Ergebnisdaten für Drucktaste

Verwendung

`btnres` (*button result*) stellt die vom Benutzer gewählte Drucktastenanzeige am Benutzergerät wie beispielsweise dem FlexPendant dar.

Beschreibung

Eine `btnres`-Konstante wird verwendet, wenn das Ergebnis der Instruktion `UIMsgBox` und der Rückgabewert der Funktionen `UIMessageBox` und `UIListView` überprüft werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `btnres`:

Beispiel 1:

```
VAR btnres answer;

UIMsgBox "More ?" \Buttons:=btnYesNo \Result:= answer;
IF answer= resYes THEN
  ...
ELSEIF answer =ResNo THEN
  ...
ENDIF
```

Die standardmäßige Tastenaufzählung `btnYesNo` bewirkt an der Benutzeroberfläche die Anzeige einer Taste "Ja" und einer Taste "Nein". Die Benutzerauswahl wird in der Variablen `answer` gespeichert.

Vordefinierte Daten

Folgende Konstanten des Datentyps `btnres` sind im System vordefiniert:

Wert	Konstanten	Tastenergebnis
0	`resUnkwn`	Unbekanntes Ergebnis
1	`resOK`	OK
2	`resAbort`	Abbrechen
3	`resRetry`	Neuer Versuch
4	`resIgnore`	Ignorieren
5	`resCancel`	Abbrechen
6	`resYes`	Ja
7	`resNo`	Nein

Es kann mit den benutzerdefinierten Drucktasten gearbeitet werden, die durch die Funktionen `UIMessageBox` und `UIListView` angezeigt werden.

Eigenschaften

`btnres` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.4 btnres - Ergebnisdaten für Drucktaste

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Benutzerinteraktions-Meldungsfenster	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion "Listenansicht"	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
Tastendaten Alias-Datentyp	*buttondata – Drucktastendaten auf Seite 1642*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.5 busstate - Status des E/A-Netzwerks

Verwendung

`busstate` wird zur Wiedergabe des aktuellen Status des E/A-Netzwerks verwendet.

Beschreibung

Eine `busstate`-Konstante dient der Prüfung des Rückgabewerts aus der Instruktion `IOBusState`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `busstate`:

Beispiel 1:

```
VAR busstate bstate;

IOBusState "IBS", bstate \Phys;
TEST bstate
CASE IOBUS_PHYS_STATE_RUNNING:
  ! Possible to access some signal on the IBS bus
DEFAULT:
  ! Actions for not up and running IBS bus
ENDTEST
```

Vordefinierte Daten

Die vordefinierten symbolischen Konstanten des Datentyps `busstate` können mit der Instruktion `IOBusState` angezeigt werden.

Eigenschaften

`busstate` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Abrufen des aktuellen Status des E/A-Netzwerks	*IOBusState - Abrufen des aktuellen Status des E/A-Netzwerks auf Seite 288*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.6 buttondata – Drucktastendaten

Verwendung

`buttondata` stellt eine standardmäßige Drucktastenkombination für die Anzeige auf dem Benutzergerät wie beispielsweise dem FlexPendant dar.

Beschreibung

Eine `buttondata`-Konstante wird für Reaktionsdrucktasten in der Instruktion `UIMsgBox` und den Funktionen `UIMessageBox` und `UIListView` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `buttondata`:

Beispiel 1:

```
VAR btnres answer;
UIMsgBox "More ?" \Buttons:=btnYesNo \Result:= answer;
IF answer= resYes THEN
...
ELSE
...
ENDIF
```

Die standardmäßige Tastenaufzählung `btnYesNo` bewirkt die Anzeige einer Taste „Ja“ und einer Taste „Nein“.

Vordefinierte Daten

Folgende Konstanten des Datentyps `buttondata` sind im System vordefiniert.

Wert	Konstanten	Angezeigte Taste
- 1	`btnNone`	Taste "Nein"
0	`btnOK`	OK
1	`btnAbrtRtryIgn`	"Abbrechen", "Neuer Versuch" und "Ignorieren"
2	`btnOKCancel`	"OK" und "Abbrechen"
3	`btnRetryCancel`	"Neuer Versuch" und "Abbrechen"
4	`btnYesNo`	"Ja" und "Nein"
5	`btnYesNoCancel`	"Ja", "Nein" und "Abbrechen"

Mithilfe der Funktionen `UIMessageBox` und `UIListView` können benutzerdefinierte Drucktasten angezeigt werden.

Eigenschaften

`buttondata` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Benutzerinteraktions-Meldungsfenster	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion "Listenansicht"	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
Tastenergebnis Alias-Datentyp	*btnres - Ergebnisdaten für Drucktaste auf Seite 1639*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.7 byte – Ganzzahlige Werte 0-255

Verwendung

`byte` wird für Dezimalwerte (0-255) gemäß dem Bereich eines Bytes verwendet.

Dieser Datentyp wird gemeinsam mit Instruktionen und Funktionen für Bitmanipulationen und Konvertierungsfunktionen verwendet.

Beschreibung

Daten des Typs `byte` stellen einen ganzzahligen Byte-Wert dar.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `byte`:

Beispiel 1:

```
VAR byte data1 := 130;
```

Definition der Variablen `data1` mit dem Dezimalwert 130.

Beispiel 2:

```
CONST num parity_bit := 8;
VAR byte data1 := 130;
BitClear data1, parity_bit;
```

Bit Nummer 8 (`parity_bit`) in der Variablen `data1` wird auf 0 gesetzt, z. B. wird der Inhalt der Variablen `data1` von 130 in 2 geändert (Ganzzahldarstellung).

Fehlerbehandlung

Wenn ein Argument des Typs `byte` einen Wert hat, der außerhalb des Bereichs zwischen 0 und 255 liegt, wird bei der Abarbeitung des Programms ein Fehler zurückgegeben.

Eigenschaften

`byte` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*
Bit-Funktionen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.8 cameradev – Kameragerät

Verwendung

`cameradev` *(Kamera)* wird verwendet, um die verschiedenen Kameras zu definieren, die vom RAPID-Programm aus gesteuert und abgerufen werden können. Der Datentyp`cameradev` wird für Instruktionen und Funktionen verwendet, die mit der Kamera kommunizieren. Die Namen der Kameras werden in den Systemparametern definiert und dürfen nicht im Programm definiert werden.

Beschreibung

Daten des Typs `cameradev` enthalten nur einen Verweis auf das Kameragerät.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `cameradev`.

Beispiel 1:

```
CamLoadJob mycamera, "myjob.job";
```

Vordefinierte Daten

Alle in den Systemparametern definierten Kameras sind in jeder Programmtask vordefiniert.

Einschränkungen

Daten des Typs `cameradev` dürfen nicht im Programm definiert werden. Anderenfalls wird eine Fehlermeldung angezeigt, sobald eine Instruktion oder Funktion abgearbeitet wird, die auf dieses `cameradev` verweist. Der Datentyp kann hingegen beim Deklarieren einer Routine als Parameter benutzt werden.

Eigenschaften

`cameradev` ist ein Datentyp ohne einen Wert (Non-Value). Das heißt, Daten dieses Typs erlauben keine wertorientierten Operationen.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.9 camerastatus - Kamera-Kommunikationsstatus

Verwendung

`camerastatus` wird für die Darstellung des Kommunikationsstatus mit der Kamera verwendet.

Beschreibung

Der Kamerastatus wird mithilfe der Funktion `CamGetMode` abgerufen und kann für die Programmablaufsteuerung oder das Debugging verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `camerastatus`:

Beispiel 1:

```
VAR camerastatus curr_camerastatus;
...
curr_camerastatus:=CamGetMode(mycamera);
IF curr_camerastatus = CAMERA_DISCON THEN
  TPWrite "Current mode of camera " + CamGetName(mycamera) + " is
       CAMERA_DISCON";
ELSEIF curr_camerastatus = CAMERA_STANDBY THEN
  TPWrite "Current mode of camera "+CamGetName(mycamera) + " is
       CAMERA_STANDBY";
ELSEIF curr_camerastatus = CAMERA_RUNNING THEN
  TPWrite "Current mode of camera " + CamGetName(mycamera) + " is
       CAMERA_RUNNING";
ENDIF
```

Aktuellen Modus der Kamera abrufen und den Status auf das FlexPendant schreiben.

Vordefinierte Daten

Folgende Konstanten des Typs `camerastatus` sind vordefiniert:

RAPID Konstante	Wert	
`CAMERA_DISCON`	1	Die Kamera ist getrennt.
`CAMERA_STANDBY`	2	Die Kamera ist im Standby-Modus.
`CAMERA_RUNNING`	3	Die Kamera läuft.

Eigenschaften

`camerastatus` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*
Kameramodus abrufen.	*CamGetMode - Aktuellen Modus der Kamera abrufen auf Seite 1241*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.10 cameratarget – Kameradaten

Verwendung

`cameratarget` wird verwendet, um Vision-Daten des Kamerabilds an das RAPID-Programm zu übergeben.

Beschreibung

Daten des Typs `cameratarget` sind eine benutzerdefinierte Sammlung von Daten, die festgelegt werden können, um Vision-Daten des Kamerabilds mit dem RAPID-Programm auszutauschen

Die Daten haben eine Reihe von Komponenten, die nach spezifischen Bedürfnissen der aktuellen Vision-Anwendung eingestellt werden können. Die Komponente `cframe` ist für die Übertragung von Informationen über die Position eines Objekts bestimmt, wohingegen die numerischen Werte und Zeichenfolgen dafür bestimmt sind, Inspektionsdaten zu enthalten.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `cameratarget`.

Beispiel 1:

```
VAR cameratarget target1;
...
wobjmycamera.oframe := target1.cframe;
MoveL pickpart, v100, fine, mygripper \WObj:= wobjmycamera;
```

Die Koordinatentransformation `cframe` wird dem Objektrahmen des Arbeitsobjekts zugewiesen. Das `robtargetpickpart` wurde zuvor auf eine richtige Entnahmeposition innerhalb des Objektrahmens des Arbeitsobjekts abgestimmt.

Komponenten

Der Datentyp hat die folgenden Komponenten:

`name`

Datentyp: `string`

Die Namensbezeichnung des `cameratarget`.

`cframe`

current frame

Datentyp: `pose`

Für die Speicherung von Positionsdaten, die normalerweise zur Führung des Roboters durch Änderung der Arbeitsobjekts verwendet wird.

`val1`

value 1

Datentyp: `num`

Für die Speicherung numerischer Ausgänge, wie z. B. Messungen.

`...`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`val5`

value 5

Datentyp: `num`

Für die Speicherung numerischer Ausgänge, wie z. B. Messungen.

`string1`

Datentyp: `string`

Für die Speicherung von numerischem optischem Ausgang, wie z. B. Inspektions- oder Identifikationsausgang.

`string2`

Datentyp: `string`

Für die Speicherung von numerischem optischem Ausgang, wie z. B. Inspektions- oder Identifikationsausgang.

`type`

Datentyp: `num`

Ein numerischer Bezeichner des Kameraziels. Gleicher Verwendungszweck wie die Komponente `name`.

`cameraname`

Datentyp: `string`

Name der Kamera

`sceneid`

scene identification

Datentyp: `num`

Der eindeutige Bezeichner des Bilds, der für die Generierung des `cameratarget` **verwendet wird.**

Struktur

```
< dataobject of cameratarget >
  < name of string >
  < cframe of pose >
    < trans of pos >
    < rot of orient >
  < val1 of num >
  < val2 of num >
  < val3 of num >
  < val4 of num >
  < val5 of num >
  < string1 of string >
  < string2 of string >
  < type of num >
  < cameraname of string >
  < sceneid of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.10 cameratarget – Kameradaten

Integrated Vision

Fortsetzung

Weitere Informationen

Information über	Siehe
Integrated Vision	*Anwendungshandbuch - Integrated Vision (integrierte Vision)*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.11 capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker

Verwendung

`capaptrreferencedata` wird verwendet, um die erforderlichen Informationen für den At-Point-Tracker-Korrekturprozess einzurichten, der durch die `CapAPTrSetupAO`, `CapAPTrSetupAI` und `CapAPTrSetupPERS`-Instruktionen eingerichtet wurde.

Komponenten

`reference_y`

Datentyp: `num`

Definiert die Referenz für die Y-Position.

`reference_z`

Datentyp: `num`

Definiert die Referenz für die Z-Position.

`threshold_y`

Datentyp: `num`

Die Differenz zwischen dem Eingangssignal und dem `reference_y` Wert muss größer als der `threshold_y` Wert sein, mit dem der Regler auf die Änderung reagiert.

`threshold_z`

Datentyp: `num`

Die Differenz zwischen dem Eingangssignal und dem `reference_z` Wert muss größer als der `threshold_z` Wert sein, mit dem der Regler auf die Änderung reagiert.

`gain_y`

Datentyp: `num`

Die Differenz zwischen dem `reference_y` Wert und dem Eingangssignalwert wird mit dem `gain_y`-Wert skaliert.

`gain_z`

Datentyp: `num`

Die Differenz zwischen dem `reference_z` Wert und dem Eingangssignalwert wird mit dem `gain_z`-Wert skaliert.

Struktur

```
< data object of capaptrreferencedata >
  < reference_y of num >
  < reference_z of num >
  < threshold_y of num >
  < threshold_z of num >
  < gain_y of num >
  < gain_z of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.11 capaptrreferencedata - Variable Setup-Daten für At-Point-Tracker

Continuous Application Platform (CAP)

Fortsetzung

Weitere Informationen

Information über	Siehe
Instruktion `CapAPTrSetupAI`	*CapAPTrSetupAI - Einen At-Point-Tracker einrichten, der von analogen Eingangssignalen gesteuert wird. auf Seite 86*
Instruktion `CapAPTrSetupAO`	*CapAPTrSetupAO - Einen At-Point-Tracker einrichten, der von analogen Ausgangssignalen gesteuert wird. auf Seite 89*
Instruktion `CapAPTrSetupPERS`	*CapAPTrSetupPERS - Einen At-Point-Tracker einrichten, der von persistenten Variablen gesteuert wird. auf Seite 92*
Continuous Application Platform	*Application manual - Continuous Application Platform*
Sensor Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.12 capdata - CAP-Daten

Verwendung

`capdata` enthält alle notwendigen Daten für die Definition das Verhaltens des CAP-Prozesses.

Komponenten

start_fly

Fliegender Start

Datentyp: `bool`

Legt fest, ob Flying Start verwendet wird oder nicht.

Wert	Folge
TRUE	Flying Start wird verwendet
FALSE	Flying Start wird NICHT verwendet

Flying Start (fliegender Start) bedeutet, dass die Roboterbewegung gestartet wird, bevor der Prozess gestartet wird. Der Prozess wird dann unterwegs gestartet (siehe *flypointdata - Daten für Flying Start/End auf Seite 1708*).

end_fly

Flying End

Datentyp: `bool`

Legt fest, ob Flying End verwendet wird oder nicht.

Wert	Folge
TRUE	Flying End wird verwendet
FALSE	Flying End wird NICHT verwendet

Flying End (fliegendes Ende) bedeutet, dass der CAP-Prozess beendet werden kann, bevor der Roboter den Endpunkt erreicht. Das ermöglicht es dem Roboter, den Prozesspfad unterwegs zu verlassen, und zwar mithilfe eines Zonenpunkts (siehe *flypointdata - Daten für Flying Start/End auf Seite 1708*).

first_instr

Erste Instruktion

Datentyp: `bool`

Definiert, ob eine `CapL`/`CapC`-Instruktion die erste Instruktion in einer Sequenz von `CapL`/`CapC`-Instruktionen ist:

Wert	Folge
TRUE	Dies ist die erste Instruktion in einer Sequenz von `CapL`/`CapC`-Instruktionen
FALSE	Dies ist nicht die erste Instruktion in einer Sequenz von `CapL`/`CapC`-Instruktionen

last_instr

Letzte Instruktion

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.12 capdata - CAP-Daten
Continuous Application Platform (CAP)
Fortsetzung

Datentyp: `bool`

Definiert, ob eine `CapL/CapC`-Instruktion die letzte Instruktion in einer Sequenz von `CapL/CapC`-Instruktionen ist:

Wert	Folge
TRUE	Dies ist die letzte Instruktion in einer Sequenz von `CapL/CapC`-Instruktionen
FALSE	Dies ist nicht die letzte Instruktion in einer Sequenz von `CapL/CapC`-Instruktionen

restart_dist

Neustart-Abstand, Einheit: mm

Datentyp: `num`

Definiert den Abstand, den der Roboter zurückgehen muss, wenn er nach einem Stopp während eines aktiven CAP-Prozesses neu gestartet wurde.

In MultiMove-Systemen müssen alle synchronisierten Roboter denselben Neustart-Abstand verwenden.

speed_data

Geschwindigkeitsdaten für CAP

Datentyp: `capspeeddata`

Definiert die gesamte CAP-Datengeschwindigkeit (siehe *capspeeddata - Geschwindigkeitsdaten für CAP auf Seite 1661*).

start_fly_point

Datentyp: `flypointdata`

Diese Daten werden nur berücksichtigt, wenn `start_fly TRUE` ist.

Definiert die Flying Start-Informationen für den CAP-Prozess (siehe *flypointdata - Daten für Flying Start/End auf Seite 1708*).

end_fly_point

Datentyp: `flypointdata`

Diese Daten werden nur berücksichtigt, wenn `end_fly TRUE` ist.

Definiert die Flying End-Informationen für den CAP-Prozess (siehe *flypointdata - Daten für Flying Start/End auf Seite 1708*).

sup_timeouts

Datentyp: `supervtimeouts`

Definiert die Zeitüberschreitungen für alle Phasen der Handshake-Überwachung (siehe *supervtimeouts - Zeitüberschreitungen der Handshake-Überwachung auf Seite 1809* und Abschnitt *Supervision* in *Application manual - Continuous Application Platform*).

proc_times

Datentyp: `processtimes`

Definiert die Zeitüberschreitungen für die Phase der Statusüberwachung PRE, POST1, und POST2 (siehe *processtimes- Prozesszeiten auf Seite 1760* und Abschnitt

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Supervision and process phases in *Application manual - Continuous Application Platform*).

block_at_restart

Datentyp: `restartblkdata`

Definiert das Verhalten des CAP-Prozesses bei einem Neustart (siehe *restartblkdata - Blockdaten für Neustart auf Seite 1766*).

Struktur

```
< data object of capdata >
  < start_fly of bool >
  < end_fly of bool >
  < first_instr of bool >
  < last_instr of bool >
  < restart_dist of num >
  < speed_data of capspeeddata >
    < fly_start of num >
    < start of num >
    < startspeed_time of num >
    < startmove_delay of num >
    < main of num >
    < fly_end of num >
  < start_fly_point of flypointdata >
    < from_start of bool >
    < time_before of num >
    < distance of num >
  < end_fly_point of flypointdata >
    < from_start of bool >
    < time_before of num >
    < distance of num >
  < sup_timeouts of supervtimeouts >
    < pre_cond of num >
    < start_cond of num >
    < end_main_cond of num >
    < end_post1_cond of num >
    < end_post2_cond of num >
  < proc_times of processtimes >
    < pre of num >
    < post1 of num >
    < post2 of num >
  < block_at_restart of restartblkdata >
    < weave_start of bool >
    < motion_delay of bool >
    < pre_phase of bool >
    < startspeed_phase of bool >
    < post1_phase of bool >
    < post2_phase of bool >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.12 capdata - CAP-Daten

Continuous Application Platform (CAP)

Fortsetzung

Weitere Informationen

	Beschrieben in:
Continuous Application Platform	*Application manual - Continuous Application Platform*
`capspeeddata` **Datentyp**	*capspeeddata - Geschwindigkeitsdaten für CAP auf Seite 1661*
`flypointdata` **Datentyp**	*flypointdata - Daten für Flying Start/End auf Seite 1708*
`supervtimeouts` **Datentyp**	*supervtimeouts - Zeitüberschreitungen der Handshake-Überwachung auf Seite 1809*
`processtimes` **Datentyp**	*processtimes- Prozesszeiten auf Seite 1760*
`block_at_restart` **Datentyp**	*restartblkdata - Blockdaten für Neustart auf Seite 1766*
Instruktion `CapL`	*CapL - Instruktion für lineare CAP-Bewegung auf Seite 109*
Instruktion `CapC`	*CapC - Instruktion für kreisförmige CAP-Bewegung auf Seite 95*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.13 caplatrackdata - Track-Daten des CAP-Look-Ahead-Trackers

Verwendung

`caplatrackdata` enthält Daten, mit denen der Benutzer beeinflussen kann, wie die `CapL/CapC`-Instruktionen die Daten der Bahnkorrektur integrieren, die ein Look-Ahead-Tracker generiert (beispielsweise Laser Tracker). `caplatrackdata` sind Teil der `captrackdata`.

Grundlegende Beispiele

```
PERS captrackdata captrack := ["laser1:",50,[1,10,1,0,0,0,0,0]]
CapL p1, v200, cd, wsd, cwd, z20, tWeldGun \Track:=captrack;
```

Komponenten

joint_no

Datentyp: `num`

Definiert den Verbindungstyp (ausgedrückt als Nummer), den die Sensorausrüstung während der Bahnkorrektur verwenden soll.

filter

Datentyp: `num`

Definiert die Zeitkonstante für den Tiefpassfilter, der auf die Bahnkorrekturen angewendet werden soll. Der Wertebereich für die Komponente reicht von 1 bis 10, wobei 1 für die schnellste Reaktion (kein Filter) auf Bahnfehler steht, die vom Sensor erkannt werden.

calibframe_no

Datentyp: `num`

Definiert, welches Kalibrierungskoordinatensystem der drei Koordinatensysteme, die in `CapLATrSetup` definiert sind, verwendet werden soll.

Wert	Kalibrierungsko-ordinatensystem	Beschreibung
1	calibframe	Obligatorisch in `CapLATrSetup`
2	calibframe2	Optional in `CapLATrSetup`
3	calibframe3	Optional in `CapLATrSetup`

seamoffs_y . seamoffs_z

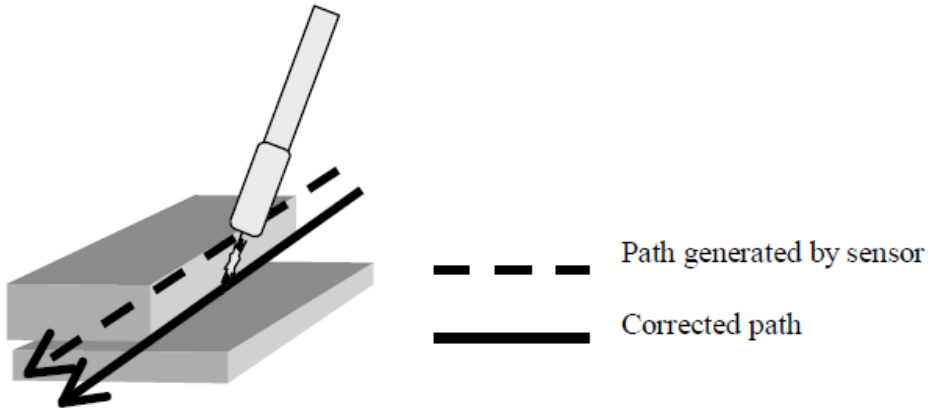
Datentyp: `num`

Mit den Komponenten für den Naht-Offset werden der Bahn Verschiebungen hinzugefügt (in mm), die vom Sensoreingang erzeugt werden. Wenn z. B die obere Kante eines Überlappstoßes vom Sensor als die korrekte Nahtposition erachtet

Fortsetzung auf nächster Seite

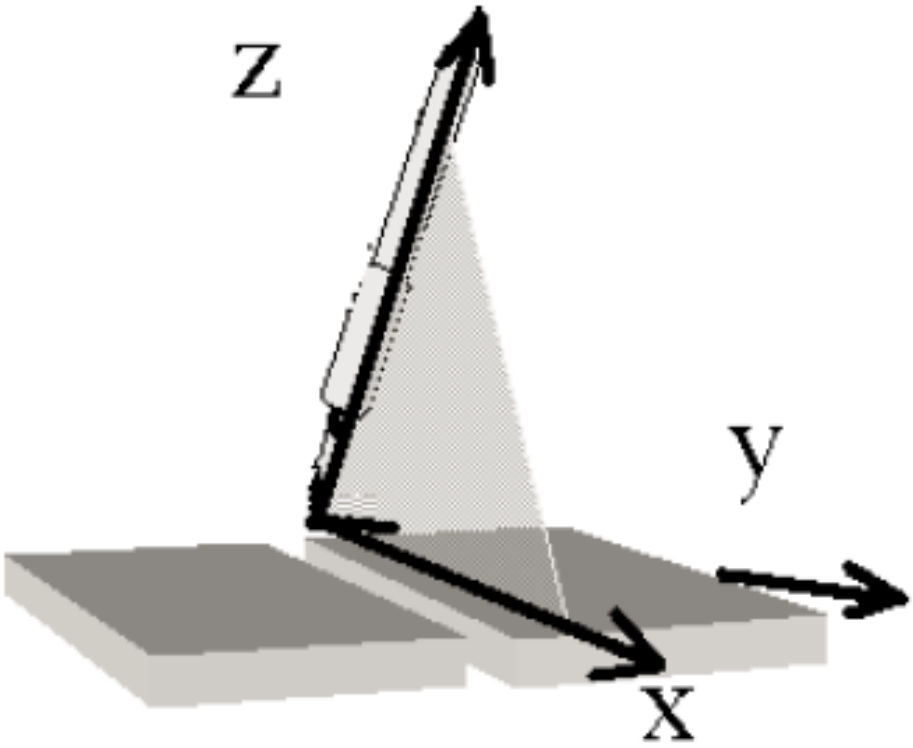
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

wird (wie in der nachfolgenden Abbildung), kann die Bahn anhand der Naht-Offsets korrigiert werden.



xx1200000199

Die Korrekturen werden im Bahnkoordinatensystem definiert, das sich rechts befindet.



xx1200000200

- Die X-Richtung verläuft parallel zur Bahntangente.
- Die Z-Richtung ist der Z-Vektor des Werkzeugs.
- Die Z-Richtung verläuft senkrecht zu einer Ebene, die durch die X- und Y-Richtungen verläuft.

seamadapt_y, seamadapt_z

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die seamadapt-Komponenten ähneln denen für den Naht-Offset. Die Offset-Größen werden jedoch nicht als feste Werte angegeben. Die Offsets werden berechnet, indem der gemessene Positionsabstand der Naht mit den seamadapt-Werten multipliziert wird.

Die Komponenten werden verwendet, um das Werkzeug hinsichtlich der Naht zu versetzen, um den Vorgang für unterschiedliche Positionsabstände zu optimieren.

track_mode

Datentyp: `num`

Mit der `track_mode`-Komponente ist es möglich, das Nachverfolgungsverhalten des Laser-Trackers selektiv zu beeinflussen.

Wert	Nachverfolgungsmodus
0	Normale Nachverfolgung. y- und z-Korrekturen werden in Betracht gezogen.
1	Die Nachverfolgung erfolgt so, als hätten die vom Laser-Tracker gesendeten y-Korrekturen den Wert Null, d. h. nur die z-Korrekturen werden in Betracht gezogen. [i]
2	Die Nachverfolgung erfolgt so, als hätten die vom Laser-Tracker gesendeten z-Korrekturen den Wert Null, d. h. nur die y-Korrekturen werden in Betracht gezogen. *i*
3	Die Nachverfolgung erfolgt so, als hätten die vom Laser-Tracker gesendeten y- und z-Korrekturen den Wert Null. *i*
4	Die y-Korrekturen werden vollständig ausgeschaltet, d. h. für die Korrekturen der y-Komponente wird Null festgelegt, bevor sie an den Roboter gesendet werden, nur die z-Korrekturen werden in Betracht gezogen. [ii]
5	Die z-Korrekturen werden vollständig ausgeschaltet, d. h. für die Korrekturen der z-Komponente wird Null festgelegt, bevor sie an den Roboter gesendet werden, nur die y-Korrekturen werden in Betracht gezogen. *ii*
6	Die y- und z-Korrekturen werden vollständig ausgeschaltet, d. h. für die Korrekturen der y- und z-Komponente wird Null festgelegt, bevor sie an den Roboter gesendet werden. *ii*
7	Die y-Korrektur wird "ausgeblendet", d. h. der TCP kehrt gekrümmt zur programmierten y-Bahnkomponente zurück. Die z-Korrektur ist aktiv.
8	Die z-Korrektur wird "ausgeblendet", d. h. der TCP kehrt gekrümmt zur programmierten z-Bahnkomponente zurück. Die y-Korrektur ist aktiv.
9	Die y- und z-Korrekturen werden "ausgeblendet", d. h. der TCP kehrt gekrümmt zur programmierten Bahnkomponente zurück.
10	Die y-Korrektur wird „eingeblendet", d. h. der TCP kehrt gekrümmt zur programmierten y-Bahnkomponente zurück. Die z-Korrektur ist aktiv.
11	Die z-Korrektur wird „eingeblendet", d. h. der TCP kehrt gekrümmt zur programmierten y-Bahnkomponente zurück. Die y-Korrektur ist aktiv.
12	Die y- und z-Korrekturen werden „eingeblendet", d. h. der TCP kehrt gekrümmt zur programmierten Bahnkomponente zurück.
13	Verfolgung, als ob von dem Laser Tracker gesendete y-Korrekturen null wären. Die Differenz zu `track_mode` 1 besteht darin, dass der Modus an der TCP-Roboterposition und nicht an der TCP-Sensorposition beginnt. *i*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.13 caplatrackdata - Track-Daten des CAP-Look-Ahead-Trackers

Continuous Application Platform (CAP)

Fortsetzung

Wert	Nachverfolgungsmodus
14	Verfolgung, als ob von dem Laser Tracker gesendete z-Korrekturen null wären. Die Differenz zu `track_mode` 2 besteht darin, dass der Modus an der TCP-Roboterposition und nicht an der TCP-Sensorposition beginnt. [i]
15	Verfolgung, als ob von dem Laser Tracker gesendete y- und z-Korrekturen null wären. Die Differenz zu `track_mode` 3 besteht darin, dass der Modus an der TCP-Roboterposition und nicht an der TCP-Sensorposition beginnt. [i]

i Bei `track_mode` 1, 2 oder 3 wird die akkumulierte Korrektur aus der vorherigen `CapL/CapC`-Instruktion für y und/oder z vorbehalten und an die nächste `CapL/CapC`-Instruktion weitergeleitet. Das ist während der gesamten Lebensdauer des CAP-Prozesses der Fall. Ein neuer CAP-Prozess wird nicht beeinflusst.

ii Für `track_mode`4, 5 oder 6 werden die Sensorwerte akkumuliert, obwohl für die y- und/oder z-Korrekturen als Wert Null festgelegt wird, bevor sie an den Roboter gesendet werden. Das bedeutet, am Anfang und Ende der Instruktion `CapL/CapC` kommt es u. U. zu einer Geschwindigkeitsverringerung.

Syntax

```
< data object of caplatrackdata >
  < joint_no of num >
  < filter of num >
  < calibframe_no of num >
  < seamoffs_y of num >
  < seamoffs_z of num >
  < seamadapt_y of num >
  < seamadapt_z of num >
  < track_mode of num >
```

Weitere Informationen

	Beschrieben in:
`captrackdata` **Datentyp**	*captrackdata - CAP-Track-Daten auf Seite 1664*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.14 capspeeddata - Geschwindigkeitsdaten für CAP

Verwendung

`capspeeddata` wird verwendet, um alle Daten eines CAP-Prozesses bezüglich der Geschwindigkeit zu definieren. Sie gehören zu den `capdata` und definieren alle Geschwindigkeitsdaten und benötigten Prozesszeiten eines CAP-Prozesses:

- Geschwindigkeit und wie lange diese Geschwindigkeit bei Beginn des CAP-Prozesses einzuhalten ist,
- Verzögerung der Bewegung des Roboters relativ zum Beginn des CAP-Prozesses,
- Geschwindigkeit des CAP-Prozesses,

Die Geschwindigkeit ist auch durch die Leistung des Roboters beschränkt. Abhängig vom Robotermodell und der Bewegungsbahn gibt es dabei Unterschiede.

Komponenten

fly_start

Datentyp: `num`

Nicht verwendet

start

Datentyp: `num`

Geschwindigkeit, die (in mm/s) zu Beginn des CAP-Prozesses verwendet wird.

startspeed_time

Datentyp: `num`

Die Zeit (in Sekunden) der `start`-Geschwindigkeit.

startmove_delay

Datentyp: `num`

Die Zeit (in Sekunden), in der die Roboterbewegung relativ zum Beginn des CAP-Prozesses verzögert wird.

main

Datentyp: `num`

Geschwindigkeit des CAP-Hauptprozesses (mm/s).

fly_end

Datentyp: `num`

Nicht verwendet

Struktur

```
< data object of capspeeddata >
  < fly_start of num >
  < start of num >
  < startspeed_time of num >
  < startmove_delay of num >
  < main of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.14 capspeeddata - Geschwindigkeitsdaten für CAP

Continuous Application Platform (CAP)

Fortsetzung

```
< fly_end of num >
```

Weitere Informationen

	Beschrieben in:
`capdata` Datentyp	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.15 capstopmode - Definiert die Stopp-Modi für CAP

Verwendung

`capstopmode` wird zur Definition der verfügbaren Stopp-Modi in CAP verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `capstopmode`.

Beispiel 1:

```
CAPSetStopMode SMOOTH_STOP_ON_PATH;
```

Vorgegebene Werte

Wert	Beschreibung
`SMOOTH_STOP_ON_PATH`	Die Roboterbewegung wird sanft gestoppt, ohne die programmierte Bahn zu verlassen.
`QUICK_STOP_ON_PATH`	Die Roboterbewegung wird so schnell wie möglich gestoppt, ohne die programmierte Bahn zu verlassen.
`EMERGENCY_STOP`	Die Roboterbewegung wird so schnell wie möglich gestoppt, gleichgültig ob er die programmierte Bahn verlässt.

Eigenschaften

`capstopmode` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

	Beschrieben in:
Instruktion `CAPSetStopMode`	*CAPSetStopMode - Stellen Sie den Stopp-Modus für Execution Error ein auf Seite 127*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.16 captrackdata - CAP-Track-Daten

Verwendung

`captrackdata` stellt die `CapL/CapC`-Instruktionen mit allen erforderlichen Daten für die Bahnkorrektur mit einem Look-Ahead- oder At-Punkt-Tracker zur Verfügung. Die Daten werden unter Verwendung des optionalen Arguments `\Track` an die `CapL/C`-Instruktionen weitergegeben.

Die Komponente `device` bestimmt, welche Art von Tracker verwendet werden soll. `captrackdata` kann nicht innerhalb einer Sequenz der `CapL/CapC`-Instruktionen geändert werden. Die Komponente `device` wird von der ersten `CapL/C`-Instruktion festgelegt. Wenn sie sich von den verbleibenden `CapL/C`-Instruktionen der gleichen Sequenz der `CapL/CapC`-Instruktionen unterscheidet, hat sie keine Wirkung.

Um die `captrackdata` zu verändern, die in einer `CapL/CapC`-Instruktion verwendet werden, muss die Sequenz zuerst beendet werden, indem die Einstellungen der Komponente `last_inst` auf `TRUE` gesetzt werden, dies muss in den `capdata` erfolgen.

Wenn `\Track` nicht in der ersten `CapL/C`-Instruktion und allen folgenden derselben Sequenz von `CapL/CapC`-Instruktionen enthalten ist, erfolgt keine Korrektur.

Grundlegende Beispiele

SIO.cfg:

```
COM_TRP:
-Name "SCOUT:" -Type "RTP1"
-Name "digi-ip:" -Type "SOCKDEV" -PhyChannel "LAN1" -RemoteAdress
     "192.168.125.5"
```

RAPID-Programm:

```
PERS captrackdata captrack1 := ["digi-ip:",50,[1,10,1,0,0,0,0,0]];
CONST string laser := "digi-ip:";
PERS pose pose1 := [[137.867,-326.31,18.5],
     [0.640984,0.766438,0.0348674,0.0223137]];
PROC main()
  VAR pos sensorPos;
  CapLATrSetup laser, pose1, pos \SensorFreq:=10 \CorrFilter:=5
       \MaxBlind:=100 \MaxIncCorr:=2;
  WriteVar laser, 6, 1;
  ! sensor ON
  CapL p1, v200, cd, wsd_event, cwd, z20, tWeldGun
       \Track:=captrack1;
  CapC p2, p3, v200, cd2, wsd, cwd, z20, tWeldGun \Track:=captrack1;
  CapL p4, v200, cd3, wsd, cwd, fine, tWeldGun \Track:=captrack1;
  WriteVar laser, 6, 0;
  ! sensor OFF
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Komponenten

device

Sensorgerät

Datentyp: `string`

Definiert, mit welchem Gerät der Sensor verbunden ist, der in den `CapL/CapC`-Instruktionen verwendet wird, um Bahnkorrekturen zu erzeugen.

max_corr

Maximal zulässige Bahnkorrektur

Datentyp: `num`

Die Komponente definiert die maximal zulässige Bahnkorrektur.

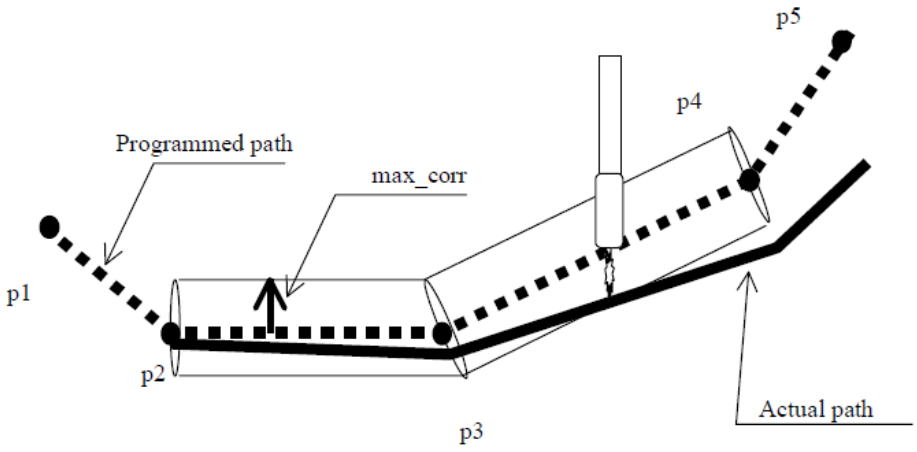
Für Look-Ahead-Tracker:

- **Wenn der TCP-Offset bei den Bahnkorrekturen größer ist als`max_corr` und `\WarnMaxCorr` in den `CapLATrSetup` eingestellt wurde, setzt der Roboter zwar seine Bahn fort, aber die übernommene Bahnkorrektur überschreitet `max_corr` nicht.**
- **Wenn `\WarnMaxCorr` nicht gesetzt wurde, wird ein Verfolgungsfehler gemeldet und die Programmabarbeitung angehalten.**

Für At-Point-Tracker:

- **Wenn der TCP-Offset infolge von Bahnkorrekturen größer ist als `max_corr`, wird ein Verfolgungsfehler gemeldet und die Programmabarbeitung wird angehalten.**

Die Abbildung zeigt das Werkzeug in einer Position relativ zur programmierten Bahn, bei der ein `max_corr` Verfolgungsfehler gemeldet würde. Einheit: mm.



xx1200000198

la_trackdata

Track-Daten des Look-Ahead-Trackers

Datentyp: `caplatrackdata`

Definiert Track-Daten, die für Look-Ahead-Tracker spezifisch sind (z.B. Laser-Tracker).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.16 captrackdata - CAP-Track-Daten

Continuous Application Platform (CAP)
Fortsetzung

Syntax

```
< data object of captrackdata >
  < device of string >
  < max_corr of num>
  < la_trackdata of caplatrackdata >
```

Weitere Informationen

	Beschrieben in:
`caplatrackdata`	*caplatrackdata - Track-Daten des CAP-Look-Ahead-Trackers auf Seite 1657*
`CapAPTrSetup`	*CapAPTrSetup - Einrichtung eines At-Point-Trackers auf Seite 83*
`CapLATrSetup`	*CapLATrSetup - Look-Ahead-Tracker einrichten auf Seite 118*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.17 capweavedata - Pendeldaten für CAP

Verwendung

`capweavedata` wird verwendet, um eine Pendelbewegung für einen CAP-Prozess während der MAIN-Phase zu definieren (siehe *Application manual - Continuous Application Platform*).

Beschreibung der Pendelbewegung

Die Pendelbewegung überlagert die Basisbahn des Prozesses. Das bedeutet, dass die Prozessgeschwindigkeit (definiert in `capspeeddata`) als definiert betrachtet wird, die TCP-Geschwindigkeit sich jedoch erhöht, bis der Roboter die physischen Grenzen erreicht.

Verfügbare Pendelbewegungstypen:

- Geometrisches Pendeln: genaueste Form
- Handgelenkpendeln: nur Roboterachse 6 wird zum Pendeln genutzt
- Schnelles Pendeln Achse 1-3: nur Roboterachse 1, 2 und 3 werden zum Pendeln genutzt
- Schnelles Pendeln Achse 4-6: nur Roboterachse 4, 5 und 6 werden zum Pendeln genutzt

Verfügbare Pendelbewegungsformen:

- Zickzack-Pendelbewegung
- V-förmige Pendelbewegung
- Dreieckige Pendelbewegung
- Kreisförmige Pendelbewegung

Alle `capweavedata`-Komponenten beziehen sich auf die MAIN-Phase.

Komponenten

Das Bahnkoordinatensystem wird definiert durch:

- X: Bahn-/Bewegungsrichtung
- Z: Z-Richtung des Werkzeugs
- Y: senkrecht sowohl zu X als auch zu Z, um ein rechtshändiges Koordinatensystem zu bilden

active

Datentyp: `bool`

Wert	Beschreibung
TRUE	Pendelbewegungen während der MAIN-Phase des CAP-Prozesses
FALSE	Führen Sie in der MAIN-Phase des CAP-Prozesses KEINE Pendelbewegungen aus

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

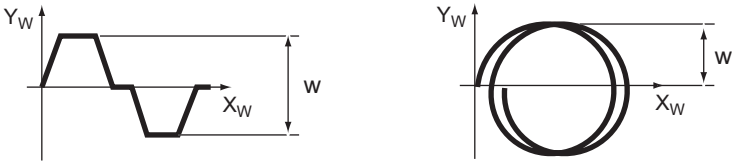
3.17 capweavedata - Pendeldaten für CAP

Continuous Application Platform (CAP)
Fortsetzung

width

Datentyp: num

Bei der kreisförmigen Pendelbewegung ist "width" der Radius des Kreises (W in der folgenden Abbildung). Bei allen anderen Pendelbewegungsarten ist "width" die Gesamtamplitude des Pendelbewegungsmusters.

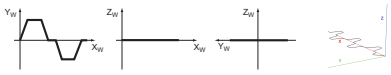
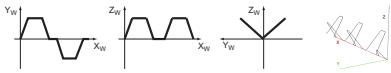
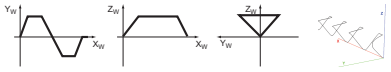
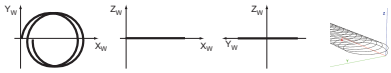


xx1200000721

shape

Datentyp: num

Die Form des Pendelbewegungsmusters in der Hauptphase.

Wert	Geometrische Form	Ergebnis
0	Keine Pendelbewegung	
1	Zickzack-Pendelbewegung	Pendelbewegung horizontal zur Naht  xx1200000714
2	V-förmige Pendelbewegung	V-förmige Pendelbewegung vertikal zur Naht  xx1200000715
3	Dreieckige Pendelbewegung	Eine dreieckige Form, vertikal zur Naht  xx1200000716
4	Kreisförmige Pendelbewegung (Nur für geometrische Pendelbewegungen verfügbar, Pendelbewegungstyp 0)	Eine kreisförmige Form, vertikal zur Naht  xx1200000717

type

Datentyp: num

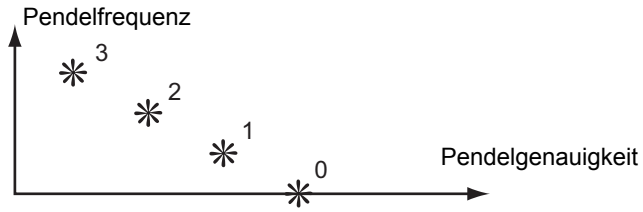
Definiert, welche Achsen während der MAIN-Phase für die Pendelbewegung genutzt werden

Angegebener Wert	Pendelart
0	Geometrisches Pendeln. Alle Achsen sind während der Pendelbewegung im Einsatz.
1	Handgelenkpendeln.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Angegebener Wert	Pendelart
2	Schnelles Pendeln. Achsen 1, 2 und 3 werden verwendet.
3	Schnelles Pendeln. Achsen 4, 5 und 6 werden verwendet.



xx1200000718

Hinweis

Alle Roboter, die TrueMove/QuickMove der zweiten Generation verwenden, weisen das folgende geänderte Verhalten für die in RW Arc und CAP verfügbaren Pendelbewegungstypen auf (im Vergleich zu TrueMove/QuickMove der ersten Generation):

- Geometrische Pendelbewegung: Keine Änderung.
- Handgelenkpendeln: Hauptsächlich Verwendung der Handgelenkachsen (4, 5 und 6), jedoch können geringe Korrekturen an den Hauptachsen vorgenommen werden, damit das Muster auf der gewünschten Ebene bleibt.
- Schnelles Pendeln: Bei TrueMove/QuickMove der zweiten Generation wurde die Leistung von geometrischem Pendeln und Handgelenkpendeln erheblich verbessert. Daher wird "Schnelles Pendeln" (beide Typen) nicht mehr als besondere Pendelart benötigt.

  Schnelles Pendeln Achse 1, 2 und 3 ist mit geometrischem Pendeln identisch.

  Schnelles Pendeln Achse 4, 5 und 6 ist mit Handgelenkpendeln identisch.

  Aus Gründen der Abwärtskompatibilität sind die Pendelarten weiterhin verfügbar.

Das System verwendet TrueMove/QuickMove der zweiten Generation, wenn es einen Schalter `dyn_ipol_type 1` in MOC.cfg in den *MOTION_PLANNER*-Daten gibt.

length

Datentyp: `num`

Definiert die Länge des Pendelzyklus in der MAIN-Phase für geometrische Pendelbewegungen (Typ = 0) und das Handgelenkpendeln (Typ = 1). Das Längenargument wird für die anderen Pendeltypen nicht verwendet.

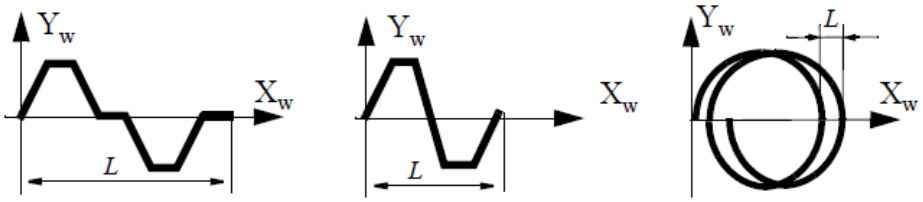
Für die kreisförmige Pendelbewegung definiert die Komponente `length` den Abstand zwischen zwei aufeinanderfolgenden Kreisen (L), wenn das `cycle_time`-Argument auf 0 gesetzt ist. Wenn das `cycle_time`-Argument einen Wert aufweist, kann das length-Attribut weggelassen werden. Bei einem positiven

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.17 capweavedata - Pendeldaten für CAP
Continuous Application Platform (CAP)
Fortsetzung

length-Wert dreht sich der TCP nach links und bei einem negativen length-Wert nach rechts.



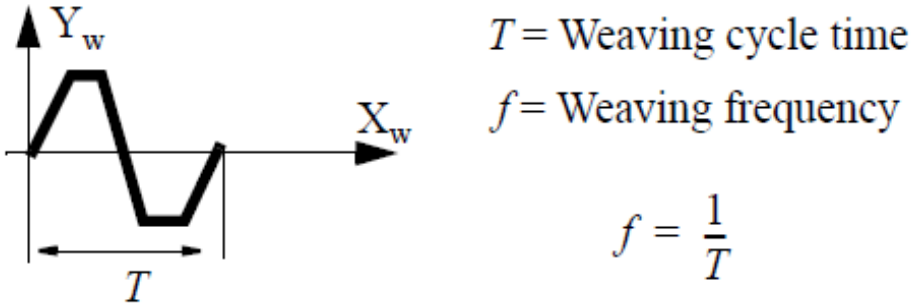
xx1200000187

cycle_time

Datentyp: `num`

Definiert die Pendelfrequenz (in Hz) in der MAIN-Phase für beide Arten des "Schnellen Pendelns" (Typ = 2 oder 3) und für das kreisförmige Pendeln. Das `cycle_time`-Argument wird für die anderen Pendelbewegungstypen nicht verwendet.

Für die kreisförmige Pendelbewegung definiert das `cycle_time` die Anzahl der Kreise pro Sekunde. Der TCP dreht sich bei einem positiven `cycle_time`-Wert nach links und bei einem negativen `cycle_time`-Wert nach rechts.



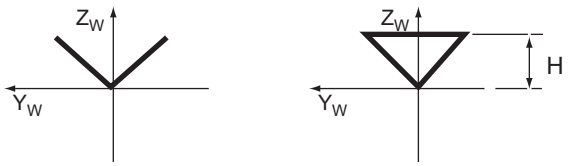
xx1200000188

height

Datentyp: `num`

Legt die Höhe des Pendelmusters (in mm) bei V-förmigen und dreieckigen Pendelbewegungen fest.

Für die kreisförmige Pendelbewegung nicht verfügbar.



xx1200000722

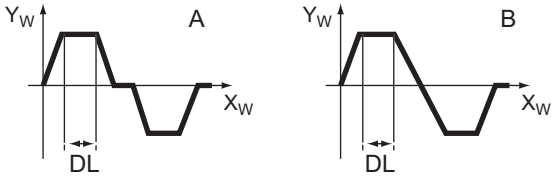
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

dwell_left

Datentyp: num

Die Länge der Verzögerung (DL), um zu erzwingen, dass sich der TCP am linken Wendepunkt der Pendelbewegung nur in Richtung der Naht bewegt. Für die kreisförmige Pendelbewegung nicht verfügbar.



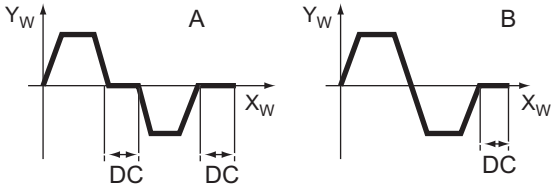
xx1200000723

A	Zickzack- und V-förmige Pendelbewegung
B	Dreieckige Pendelbewegung

dwell_center

Datentyp: num

Die Länge der Verzögerung (DC), um zu erzwingen, dass sich der TCP am Mittelpunkt der Pendelbewegung nur in Richtung der Naht bewegt.



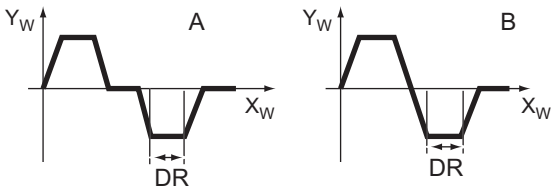
xx1200000724

A	Zickzack- und V-förmige Pendelbewegung
B	Dreieckige Pendelbewegung

dwell_right

Datentyp: num

Die Länge der Verzögerung (DR), um zu erzwingen, dass sich der TCP am rechten Wendepunkt der Pendelbewegung nur in Richtung der Naht bewegt. Für die kreisförmige Pendelbewegung nicht verfügbar.



xx1200000725

A	Zickzack- und V-förmige Pendelbewegung
B	Dreieckige Pendelbewegung

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.17 capweavedata - Pendeldaten für CAP

Continuous Application Platform (CAP)
Fortsetzung

dir

Datentyp: num

Der Richtungswinkel der Pendelbewegung horizontal zur Naht. Ein Winkel von 0 Grad führt zu einer Pendelbewegung vertikal zur Naht.

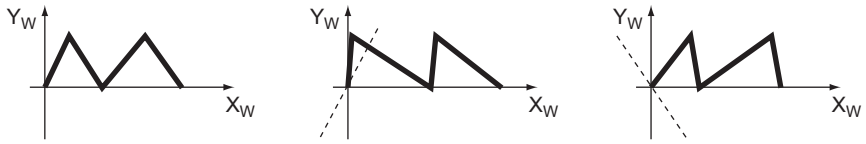


xx1200000726

tilt

Datentyp: num

Der Neigungswinkel der Pendelbewegung vertikal zur Naht. Ein Winkel von 0 Grad führt zu einer Pendelbewegung, die sich vertikal zur Naht bewegt.

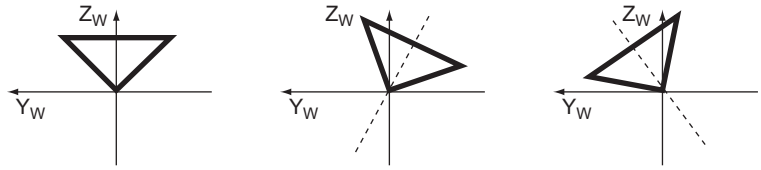


xx1200000727

rot

Datentyp: num

Der Ausrichtungswinkel der Pendelbewegung horizontal-vertikal zur Naht. Bei einem Winkel von 0 Grad wird eine symmetrische Pendelbewegung ausgeführt.



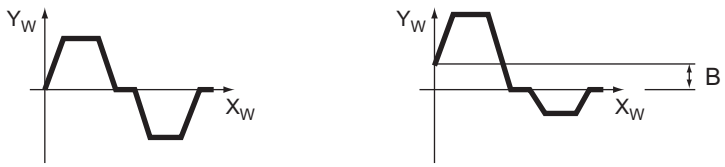
xx1200000728

bias

Datentyp: num

Die horizontale Ausrichtung des Pendelbewegungsmusters. Die Ausrichtung kann nur für eine Zickzack-Pendelbewegung festgelegt werden und darf dabei nicht größer als die halbe Breite der Pendelbewegung sein. Für die kreisförmige Pendelbewegung nicht verfügbar.

Die folgende Abbildung zeigt Zickzack-Pendelbewegungen mit und ohne Ausrichtung (B).



xx1200000729

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ptrn_sync_on

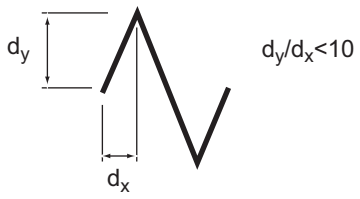
Datentyp: `bool`

Wert	Beschreibung
TRUE	Synchronisationsimpulse am rechten und linken Wendepunkt des Pendelmusters senden
FALSE	KEINE Synchronisationsimpulse am rechten und linken Wendepunkt des Pendelmusters senden

Einschränkungen

Die maximale Pendelfrequenz beträgt 2 Hz.

Die Neigung des Pendelmusters darf nicht das Verhältnis 1:10 (84 Grad) überschreiten. Siehe folgende Abbildung.



xx1200000730

Das Ändern von `weave_type` in `weavedata` ist in Zonenpunkten nicht möglich, nur in Feinpunkten. Das Verhalten ist für *TrueMove* und *QuickMove* gleich, sowohl in der ersten als auch in der zweiten Generation.

Alle Roboter, die *TrueMove* oder *QuickMove* der zweiten Generation verwenden, weisen das folgende geänderte Verhalten für die in RW Arc verfügbaren Pendelarten auf (im Vergleich zu *TrueMove* oder *QuickMove* der ersten Generation):

- Geometrisches Pendeln - keine Änderung.
- Handgelenkpendeln – hauptsächliche Verwendung der Handgelenkachsen (4, 5 und 6), jedoch können geringe Korrekturen an den Hauptachsen vorgenommen werden, damit das Muster auf der gewünschten Ebene bleibt.
- Schnelles Pendeln - bei *TrueMove* oder *QuickMove* der zweiten Generation wurde die Leistung von geometrischem Pendeln und Handgelenkpendeln erheblich verbessert. Daher wird schnelles Pendeln (beide Arten) nicht mehr als besondere Pendelart benötigt.

  Schnelles Pendeln Achse 1, 2 und 3 ist mit geometrischem Pendeln identisch.

  Schnelles Pendeln Achse 4, 5 und 6 ist mit Handgelenkpendeln identisch.

  Aus Gründen der Abwärtskompatibilität sind die Pendelarten weiterhin verfügbar.

Das System verwendet *TrueMove* oder `dyn_ipol_type 1` der zweiten Generation, wenn MOC.cfg in den *QuickMove*-Daten den Schalter MOTION_PLANNER enthält.

Syntax

```
< data object of capweavedata >
  < active of bool>
  < width of num >
  < shape of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.17 capweavedata - Pendeldaten für CAP

Continuous Application Platform (CAP)

Fortsetzung

```
< type of num >
< length of num >
< cycle_time of num >
< height of num >
< dwell_left of num >
< dwell_center of num >
< dwell_right of num >
< dir of num >
< tilt of num >
< rot of num >
< bias of num >
< ptrn_sync_on of bool >
```

Weitere Informationen

	Beschrieben in:
`capdata` Datentyp	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.18 cfgdomain - Konfigurationsdomain

Verwendung

`cfgdomain` (*Konfigurationsdomain*) wird verwendet, um eine Konfigurationsdomain anzugeben.

Beschreibung

Daten des Typs `cfgdomain` sollen zur Definition der Konfigurationsdomain dienen, die mit der Instruktion `SaveCfgData` gespeichert werden soll.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `cfgdomain`:

Beispiel 1:

```
SaveCfgData "SYSPAR" \File:="MYEIO.cfg", EIO_DOMAIN;
```

Speichert die E/A-Konfigurationsdomain in der Datei `MYEIO.cfg` im Verzeichnis SYSPAR.

Vordefinierte Daten

Die folgenden vordefinierten Konstanten können für die Angabe einer Konfigurationsdomain benutzt werden.

Name	Beschreibung
`EIO_DOMAIN`	Konfiguration des E/A-Systems
`MOC_DOMAIN`	Bewegungskonfiguration
`SIO_DOMAIN`	Kommunikationsdomain
`PROC_DOMAIN`	Prozessdomain
`SYS_DOMAIN`	Steuerungsdomain
`MMC_DOMAIN`	Mensch-Maschine-Kommunikation
`ALL_DOMAINS`	Alle oben aufgeführten Domains

Eigenschaften

`cfgdomain` ist ein Alias-Datentyp für `string` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Speichert Systemparameter in einer Datei	*SaveCfgData - Speichert Systemparameter in einer Datei auf Seite 656*
Systemparameter	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.19 clock – Zeitmessung

Verwendung

`Clock` wird zur Zeitmessung verwendet. Der Datentyp "clock" funktioniert wie eine Stoppuhr.

Beschreibung

Daten des Typs `clock` speichern eine gemessene Zeit in Sekunden in einer Auflösung von 0,001 Sekunden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `clock`:

Beispiel 1:

```
VAR clock myclock;
ClkReset myclock;
```

Die Uhr `myclock,` wird deklariert und zurückgesetzt. Bevor Sie `ClkReset`, `ClkStart`, `ClkStop` und `ClkRead` verwenden können, müssen Sie in Ihrem Programm eine Variable mit dem Datentyp `clock` deklarieren.

Einschränkungen

Die höchste Dauer, die in einer clock-Variablen gespeichert werden kann, beträgt ungefähr 49 Tage (4.294.967 Sekunden). Die Instruktionen `ClkStart`, `ClkStop` und `ClkRead` melden das Überlaufen einer Uhr, was extrem selten vorkommt.

Eine Uhr muss als Variablentyp `VAR` definiert werden, nicht als ein `persistent` Variablentyp.

Eigenschaften

`clock` ist ein Datentyp ohne einen Wert (Non-Value) und kann nicht in wertorientierten Operationen verwendet werden.

Weitere Informationen

Information über	Siehe
Zusammenfassung von Zeit- und Datums-instruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - System & Time*
Eigenschaften für Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.20 confdata – Konfigurationsdaten des Roboters

Verwendung

`confdata` wird zur Definition der Achsenkonfigurationen des Roboters verwendet.

Beschreibung

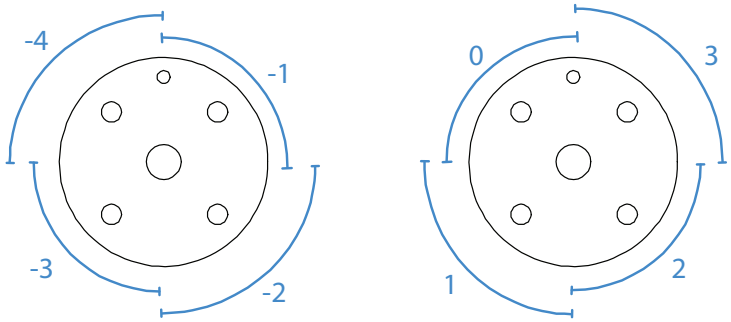
Alle Positionen des Roboters werden mithilfe von kartesischen Koordinaten definiert und gespeichert. Bei der Berechnung der entsprechenden Achsenpositionen gibt es häufig zwei oder mehr mögliche Lösungen. Das heißt, der Roboter kann bei verschiedenen Positionen oder Konfigurationen der Roboterachsen dieselbe Position erreichen, sodass das Werkzeug dieselbe Stellung und dieselbe Orientierung einnimmt.

Einige Robotertypen verwenden iterative numerische Methoden, um die Positionen der Roboterachsen zu bestimmen. In diesen Fällen lassen sich mithilfe der Konfigurationsparameter gute Anfangswerte für die Achsen definieren, die dann von der iterativen Prozedur verwendet werden.

Um eine dieser möglichen Konfigurationen eindeutig zu kennzeichnen, wird die Roboterkonfiguration mit vier Achsenwerten angegeben. Bei einer rotierenden Achse definiert der Wert den aktuellen Quadranten der Roboterachse. Die Quadranten sind mit 0, 1, 2 usw. nummeriert (sie können auch negativ sein). Die Quadrantennummer ist mit dem aktuellen Gelenkwinkel der Achse verbunden.

Bei 6-Achsen-Robotern ist Quadrant 0 die erste Viertelumdrehung von 0° bis 90° in positiver Richtung ab der Nullposition; Quadrant 1 ist die nächste Umdrehung von 90° bis 180° usw. Quadrant -1 ist die Viertelumdrehung von 0° bis (-90°) usw.

Die Abbildung zeigt die Konfigurationsquadranten für die Achsen 1, 4 oder 6 an einem 6-Achsen-Roboter, bei dem die Nullposition gerade ist.



xx0500002398

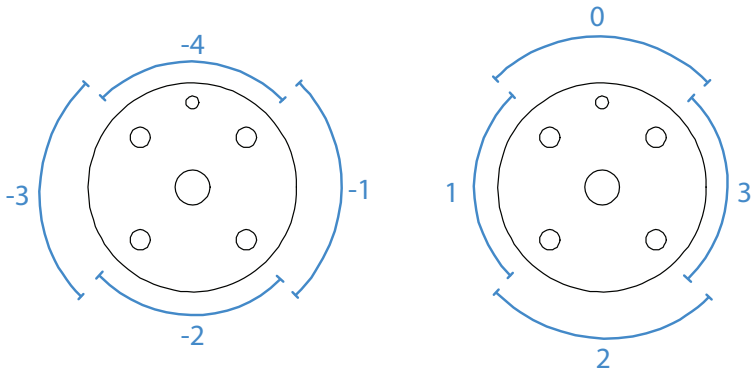
Bei 7-Achsen-Robotern ist Quadrant 0 die Viertelumdrehung um die Nullposition herum von -45° bis +45°; Quadrant 1 ist die nächste Viertelumdrehung in positiver Richtung von 45° bis 135° usw. Quadrant -1 ist die Viertelumdrehung von -45° bis -135° usw.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.20 confdata – Konfigurationsdaten des Roboters
RobotWare - OS
Fortsetzung

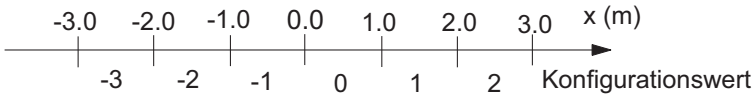
Die Abbildung zeigt die Konfigurationsquadranten für die Achsen 1, 4 oder 6 an einem 7-Achsen-Roboter, bei dem die Nullposition gerade ist.



xx1700001572

Bei einer linearen Achse definiert der Wert ein Intervall in Metern für die Roboterachse. Für jede Achse bedeutet der Wert 0 eine Position zwischen 0 und 1 Meter; 1 bedeutet eine Position zwischen 1 und 2 Metern. Bei negativen Werten bedeutet -1 eine Position zwischen -1 und 0 Metern usw.

Die Abbildung zeigt Konfigurationswerte für eine lineare Achse.



xx0500002399

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `confdata`:

Beispiel 1:

```
VAR confdata conf15 := [1, -1, 0, 0]
```

Eine Roboterkonfiguration `conf15` für einen Lackierrobotertyp wird wie folgt definiert:

- Die Achsenkonfiguration der Roboterachse 1 ist Quadrant 1, d. h. 90-180º.
- Die Achsenkonfiguration der Roboterachse 4 ist Quadrant `-1`, d. h. 0-(-90º).
- Die Achsenkonfiguration der Roboterachse 6 ist Quadrant `0`, d. h. 0 - 90º.
- Die Achsenkonfiguration der Roboterachse 5 ist Quadrant `0`, d. h. 0 - 90º.

Überwachung der Konfiguration

Bei manchen Robotermodellen werden die Konfigurationsdaten (`confdata`) auch verwendet, um eine Überwachung der programmierten Punkte bei Linearbewegungen vorzunehmen, wenn `ConfL\On` eingestellt ist.

Mit `ConfJ\On` erfolgt keine Konfigurationsüberwachung, weitere Informationen siehe *ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung auf Seite 156*.

Bevor eine angeforderte Bewegung begonnen wird, wird eine Überprüfung vorgenommen, um festzustellen, ob es möglich ist, die programmierte Konfiguration zu erreichen. Wenn dies nicht möglich ist, wird das Programm angehalten. Wenn

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

die Bewegung abgeschlossen ist (in einer Zone oder Feinpunkt), wird auch überprüft, ob der Roboter die programmierte Konfiguration erreicht hat.

Die Überwachung der Konfiguration mit `ConfL\On` läuft je nach Roboter unterschiedlich ab. Details finden Sie in den folgenden Abschnitten.

6-Achsen-Roboter

Die Konfigurationsüberwachung überprüft, ob sich die Achsen 1, 4 und 6 nicht weiter als 180 Grad bewegen und ob die angeforderte Bewegung keine Veränderung bei `cfx` erforderlich macht.

`cfx` wird standardmäßig nur von Robotern mit serieller Verbindung verwendet, nicht für Parallelroboter. Für Roboter mit Parallelstangen kann `cfx` durch Setzen des Systemparameters *Use cfx in robtargets for P-rod robots* auf *Yes* aktiviert werden.

4-Achsen-Roboter

Die Überwachung der Konfiguration wird überprüfen, dass sich die Achsen 1 und 6 nicht weiter als 180 Grad bewegen.

Parallelarmroboter (Delta-Roboter)

Die Überwachung der Konfiguration wird überprüfen, dass sich die Achse 4 nicht weiter als 180 Grad bewegt.

SCARA-Roboter

Die Überwachung der Konfiguration wird überprüfen, dass sich die Achsen 1 und 4 nicht weiter als 180 Grad bewegen. Sie prüft auch das Vorzeichen von Achse 2.

7-Achsen-Roboter

Die Konfigurationsüberwachung überprüft, ob sich die Achsen 1, 4 und 6 nicht weiter als 180 Grad bewegen und ob die angeforderte Bewegung keine Veränderung bei `cfx` erforderlich macht.

Lackierroboter

Keine Konfigurationsüberwachung wird ausgeführt.

Konfigurationsdaten des Roboters

6-Achsen-Roboter mit serieller Verbindung

Innerhalb des Arbeitsbereichs des Roboters gibt es drei Singularitäten. Weitere Informationen zu Singularitäten erhalten Sie unter *Technisches Referenzhandbuch - RAPID Overview*.

- `cf1` ist die Nummer des Quadranten für Achse 1.
- `cf4` ist die Nummer des Quadranten für Achse 4.
- `cf6` ist die Nummer des Quadranten für Achse 6.

Mit `cfx` wird eine der acht möglichen Roboterkonfigurationen gewählt, die von 0 bis 7 nummeriert sind. Folgende Tabelle beschreibt jede dieser Konfigurationen hinsichtlich der Roboterposition in Relation zu den drei Singularitäten

cfx	Mitte des Handgelenks relativ zu Achse 1	Mitte des Handgelenks relativ zu Unterarm	Winkel von Achse 5
0	Davor	Davor	Positiv

Fortsetzung auf nächster Seite

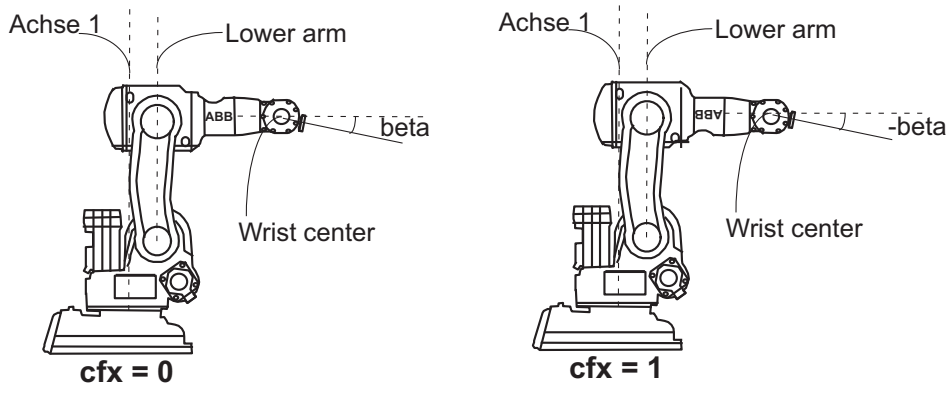
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.20 confdata – Konfigurationsdaten des Roboters
RobotWare - OS
Fortsetzung

cfx	Mitte des Handgelenks relativ zu Achse 1	Mitte des Handgelenks relativ zu Unterarm	Winkel von Achse 5
1	Davor	Davor	Negativ
2	Davor	Dahinter	Positiv
3	Davor	Dahinter	Negativ
4	Dahinter	Davor	Positiv
5	Dahinter	Davor	Negativ
6	Dahinter	Dahinter	Positiv
7	Dahinter	Dahinter	Negativ

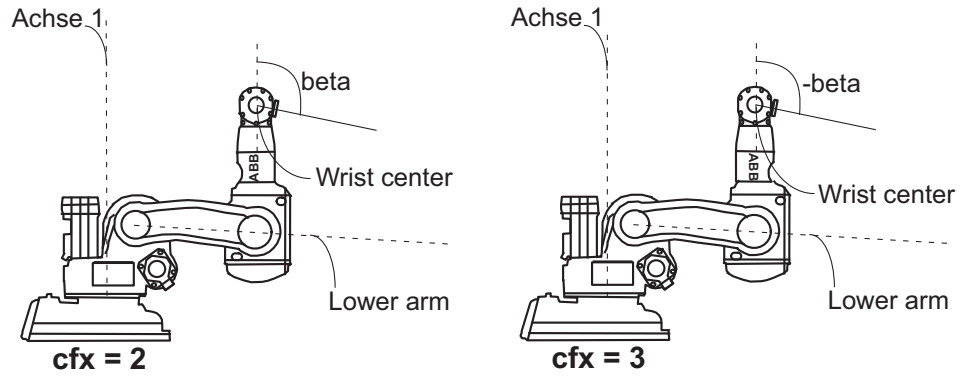
Die folgenden Abbildungen stellen die acht verschiedenen Konfigurationen mit der gleichen Werkzeugposition und -orientierung dar.

Die folgende Abbildung zeigt das Beispiel einer Roboterkonfiguration 0 und 1. Beachten Sie, dass sich wegen des veränderten Vorzeichens von Achse 5 auch cfx ändert, und man vermeiden sollte ModPos von robtargets mit Achse 5 nahe bei Null zu erzeugen oder herbeizuführen. Mit Achse 5 an Null kann das Messrauschen dazu führen, dass das Zeichen der Messposition von Achse 5 schwankt, wodurch ein unvorhersehbarer Wert cfx ausgegeben wird.



xx0500002400

Die folgende Abbildung zeigt ein Beispiel für Roboterkonfiguration 2 und 3. Beachten Sie die verschiedenen Vorzeichen des Winkels von Achse 5.

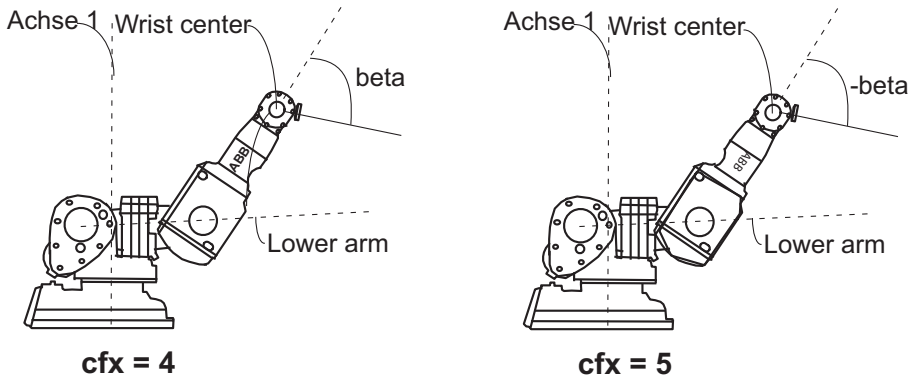


xx0500002401

Fortsetzung auf nächster Seite

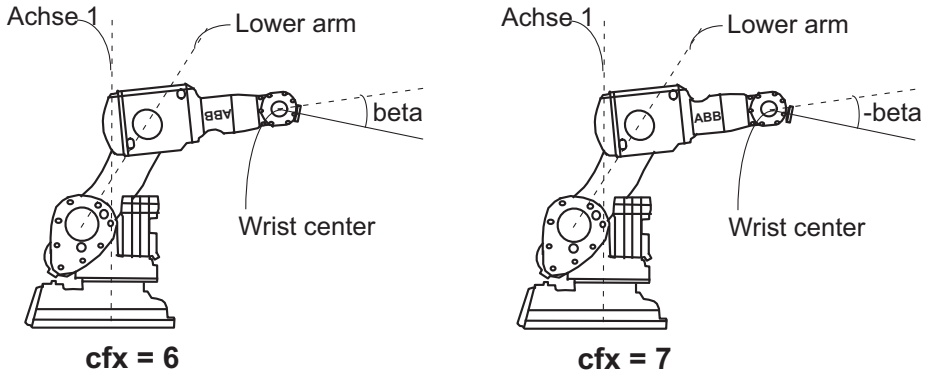
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die folgende Abbildung zeigt ein Beispiel für Roboterkonfiguration 4 und 5. Beachten Sie die verschiedenen Vorzeichen des Winkels von Achse 5.



xx0500002402

Die folgende Abbildung zeigt ein Beispiel für Roboterkonfiguration 6 und 7. Beachten Sie die verschiedenen Vorzeichen des Winkels von Achse 5.



xx0500002403

6-Achsen-Roboter mit Parallelstange

Es werden nur die drei Konfigurationsparameter `cf1`, `cf4` und `cf6` verwendet, bis der Systemparameter *Use cfx in robtargets for P-rod robots* auf *Yes* gesetzt ist.

4-Achsen-Roboter

Nur der Konfigurationsparameter `cf6` wird verwendet.

Parallelarmroboter

Nur der Konfigurationsparameter `cf4` wird verwendet.

SCARA-Roboter

Nur die drei Konfigurationsparameter `cf1`, `cf4` und `cfx` werden verwendet.

Der `cfx`-Wert wird verwendet, um das Vorzeichen des Winkels der Achse 2 anzuzeigen. `cfx` ist 1, wenn der Winkel von Achse 2 negativ ist, anderenfalls ist `cfx` 0.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.20 confdata – Konfigurationsdaten des Roboters
RobotWare - OS
Fortsetzung

7-Achsen-Roboter

Alle vier Konfigurationsparameter werden verwendet. `cf1`, `cf4`, `cf6` für Gelenke 1, 4 bzw. 6. `cfx` wird verwendet, um eine von 8 möglichen Roboterkonfigurationen zu wählen. Die Funktion ähnelt der anderer Roboter.

cfx	Achse 2 Winkel	Mitte des Handgelenks relativ zu Unterarm	Achse 5 Winkel
0	Positiv	Davor	Positiv
1	Positiv	Davor	Negativ
2	Positiv	Dahinter	Positiv
3	Positiv	Dahinter	Negativ
4	Negativ	Davor	Positiv
5	Negativ	Davor	Negativ
6	Negativ	Dahinter	Positiv
7	Negativ	Dahinter	Negativ

Lackierroboter

Alle vier Konfigurationsparameter werden verwendet, `cf1`, `cf4`, `cf6` für die Achsen 1, 4 bzw. 6 und `cfx` für Achse 5.

IRB 5500

Es werden alle vier Konfigurationsparameter genutzt. `cf1`, `cf4`, `cf6` für Gelenke 1, 4 bzw. 6. Der Parameter `cfx` enthält eine Kombination aus den Quadrantennummer von Gelenk 5 und den vier möglichen Konfigurationen für die Achsen 2 und 3.

Weitere Informationen finden Sie in *Product Manual - IRB 5500*.

IRB 5350

Die Roboter verfügen über zwei Rotationsachsen (Arm 1 und 2) und eine lineare Achse (Arm 3).

- `cf1` wird für Rotationsachse 1 verwendet.
- `cfx` wird für Rotationsachse 2 verwendet.
- `cf4` und `cf6` werden nicht verwendet.

Komponenten

`cf1`

Datentyp: `num`

Rotationsachse:

Der aktuelle Quadrant von Achse 1, ausgedrückt als positive oder negative Ganzzahl.

Lineare Achse:

Das aktuelle Intervall in Metern von Achse 1, ausgedrückt als positive oder negative Ganzzahl.

`cf4`

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Rotationsachse:

Der aktuelle Quadrant von Achse 4, ausgedrückt als positive oder negative Ganzzahl.

Lineare Achse:

Das aktuelle Intervall in Metern von Achse 4, ausgedrückt als positive oder negative Ganzzahl.

`cf6`

Datentyp: `num`

Rotationsachse:

Der aktuelle Quadrant von Achse 6, ausgedrückt als positive oder negative Ganzzahl.

Lineare Achse:

Das aktuelle Intervall in Metern von Achse 6, ausgedrückt als positive oder negative Ganzzahl.

`cfx`

Datentyp: `num`

Rotationsachse:

Bei Robotern mit serieller Verbindung die aktuelle Roboterkonfiguration, ausgedrückt als Ganzzahl im Bereich von 0 bis 7.

Bei SCARA-Robotern wird die aktuelle Roboterkonfiguration als ganze Zahl im Bereich von 0 bis 1 ausgedrückt, siehe *SCARA-Roboter auf Seite 1681*.

Für 7-Achsen-Robotern, die aktuelle Roboterkonfiguration, ausgedrückt als Ganzzahl im Bereich von 0 bis 7, siehe *7-Achsen-Roboter auf Seite 1682*.

Bei Lackierrobotern der aktuelle Quadrant von Achse 5, ausgedrückt als positive oder negative Ganzzahl. Für IRB 5500 siehe *IRB 5500 auf Seite 1682*.

Für alle anderen Robotermodelle der aktuelle Quadrant von Achse 2, ausgedrückt als positive oder negative Ganzzahl.

Lineare Achse:

Das aktuelle Intervall in Metern von Achse 2, ausgedrückt als positive oder negative Ganzzahl.

Struktur

```
< dataobject of confdata >
  < cf1 of num >
  < cf4 of num >
  < cf6 of num >
  < cfx of num >
```

Weitere Informationen

Information über	Siehe
Koordinatensysteme Handhabung von Konfigurationsdaten Singularitäten	*Technisches Referenzhandbuch - RAPID Overview*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.20 confdata – Konfigurationsdaten des Roboters

RobotWare - OS

Fortsetzung

Information über	Siehe
Positionsdaten	*robtarget - Positionsdaten auf Seite 1778*
Systemparameter	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.21 corrdescr – Deskriptor des Korrekturgenerators

Verwendung

`corrdescr` (*Correction generator descriptor*) wird von Korrekturgeneratoren verwendet. Ein Korrekturgenerator fügt dem Bahn-Koordinatensystem geometrische Offsets hinzu.

Beschreibung

Daten des Typs `corrdescr` enthalten einen Verweis auf einen Korrekturgenerator.

Durch die Anweisung `CorrCon` wird eine Verbindung zu einem Korrekturgenerator aufgebaut und der Deskriptor (der Verweis auf den Korrekturgenerator) kann für die Wiedergabe geometrischer Offsets im Bahn-Koordinatensystem mit der Anweisung `CorrWrite` verwendet werden.

Vorhergehende Offsets können durch die Unterbrechung eines Korrekturgenerators über die Anweisung `CorrDiscon` entfernt werden. Alle aktiven Korrekturgeneratoren können über die Anweisung `CorrClear` entfernt werden.

Die Funktion `CorrRead` gibt die Summe aller bisher gelieferten Offsets zurück (inklusive aller aktiven Korrekturgeneratoren).

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `corrdescr`:

Beispiel 1:

```
VAR corrdescr id;
VAR pos offset;
...
CorrCon id;
offset := [1, 2 ,3];
CorrWrite id, offset;
```

Ein Korrekturgenerator wird über die Anweisung `CorrCon` verbunden und vom Deskriptor `id` angesprochen. Dann werden Offsets über die Anweisung `CorrWrite` an den Korrekturgenerator weitergegeben (mit Verweis `id`).

Eigenschaften

`corrdescr` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Aufbauen einer Verbindung zu einem Korrekturgenerator	*CorrCon - Aufbauen einer Verbindung zu einem Korrekturgenerator auf Seite 177*
Trennen der Verbindung zu einem Korrekturgenerator	*CorrDiscon - Trennen der Verbindung zu einem Korrekturgenerator auf Seite 182*
Schreiben an einen Korrekturgenerator	*CorrWrite - Schreiben an einen Korrekturgenerator auf Seite 183*
Lesen der aktuellen Gesamt-Offsets	*CorrRead - Lesen der aktuellen Gesamt-Offsets auf Seite 1252*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.21 corrdescr – Deskriptor des Korrekturgenerators

Path Offset

Fortsetzung

Information über	Siehe
Entfernen aller Korrekturgeneratoren	*CorrClear - Entfernen aller Korrekturgeneratoren auf Seite 176*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.22 datapos – Umschließender Block für ein Datenobjekt

Verwendung

`datapos` ist der Block, in den ein Datenobjekt (interne Systemdaten) eingeschlossen wird, das mit der Funktion `GetNextSym` abgerufen wird.

Beschreibung

Daten des Typs `datapos` enthalten Informationen darüber, wo ein bestimmtes Objekt im System definiert ist. Dieser Typ wird für die Instruktionen `GetDataVal` und `SetDataVal` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `datapos`:

Beispiel 1:

```
VAR datapos block;
VAR string name;
VAR bool truevar:=TRUE;
...
SetDataSearch "bool" \Object:="my.*" \InMod:="mymod"\LocalSym;
WHILE GetNextSym(name,block) DO
  SetDataVal name\Block:=block,truevar;
ENDWHILE
```

Dieser Vorgang setzt alle lokalen `bool`-Datenobjekte, die mit `my` beginnen, im Modul `mymod` auf `TRUE`.

Eigenschaften

`datapos` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Definieren eines Symbols, das in einem Suchvorgang gesetzt wird	*SetDataSearch - Definieren eines Symbols, das in einer Suchsequenz gesetzt wird auf Seite 702*
Abrufen des nächsten passenden Symbols	*GetNextSym - Abrufen des nächsten passenden Symbols auf Seite 1328*
Abrufen des Werts eines Datenobjekts	*GetDataVal - Abrufen des Werts eines Datenobjekts auf Seite 228*
Setzen des Werts eines Datenobjekts	*SetDataVal - Setzen des Werts eines Datenobjekts auf Seite 707*
Setzen des Werts vieler Objekte	*SetAllDataVal - Setzen eines Werts für alle Datenobjekte in einer definierten Menge auf Seite 697*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.23 dionum – Digitale Werte (0-1)

Verwendung

`dionum` *(digital input output numeric)* wird für digitale Werte (0 oder 1) verwendet.

Dieser Datentyp wird gemeinsam mit Instruktionen und Funktionen zur Handhabung von digitalen Eingangs- oder Ausgangssignalen verwendet.

Beschreibung

Daten des Typs `dionum` stellen einen digitalen Wert (0 oder 1) dar.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `dionum`:

Beispiel 1:

```
CONST dionum close := 1;
SetDO grip1, close;
```

Definition der Konstanten `close` mit einem Wert gleich `1`. Das Signal `grip1` wird dann auf `close`, d. h. `1` gesetzt.

Vordefinierte Daten

Die Konstanten `high`, `low`, und `edge` sind im System vordefiniert:

```
CONST dionum low:=0;
CONST dionum high:=1;
CONST dionum edge:=2;
```

Die Konstanten `low` und `high` sind für E/A-Instruktionen vorgesehen.

`Edge` kann mit den Interrupt-Instruktionen `ISignalDI` und `ISignalDO` benutzt werden.

Eigenschaften

`dionum` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Zusammenfassung von Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and output signals*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*
Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics- Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.24 dir – Dateiverzeichnisstruktur

Verwendung

`dir` (*directory*) wird zum Durchlaufen von Verzeichnisstrukturen verwendet.

Beschreibung

Daten des Typs `dir` enthalten einen Verweis auf ein Verzeichnis auf einem Datenträger oder in einem Netzwerk. Er kann durch die Instruktion `OpenDir` mit einem physischen Verzeichnis verbunden und dann zum Lesen verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `dir`:

Beispiel 1:

```
PROC lsdir(string dirname)
  VAR dir directory;
  VAR string filename;
  OpenDir directory, dirname;
  WHILE ReadDir(directory, filename) DO
    TPWrite filename;
  ENDWHILE
  CloseDir directory;
ENDPROC
```

Dieses Beispiel gibt die Namen aller Dateien bzw. Unterverzeichnisse unter dem angegebenen Verzeichnis aus.

Eigenschaften

`dir` ist ein Datentyp ohne einen Wert (Non-Value) und kann nicht in wertorientierten Operationen verwendet werden.

Weitere Informationen

Information über	Siehe
Öffnen eines Verzeichnisses	*OpenDir - Öffnen eines Verzeichnisses auf Seite 497*
Erstellen eines Verzeichnisses	*MakeDir - Erstellen eines neuen Verzeichnisses auf Seite 357*
Lesen eines Verzeichnisses	*ReadDir - Lesen des nächsten Eintrags in einem Verzeichnis auf Seite 1459*
Schließen eines Verzeichnisses	*CloseDir - Schließen eines Verzeichnisses auf Seite 153*
Löschen eines Verzeichnisses	*RemoveDir - Löschen eines Verzeichnisses auf Seite 603*
Löschen einer Datei	*RemoveFile - Löschen einer Datei auf Seite 605*
Umbenennen einer Datei	*RenameFile - Umbenennen einer Datei auf Seite 609*
Prüfen eines Dateityps	*IsFile - Prüfen des Dateityps auf Seite 1372*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.25 dnum – Doppelte numerische Werte

Verwendung

`dnum` wird für numerische Werte wie z. B. Zähler verwendet. Der Datentyp kann größere ganzzahlige Werte als der Datentyp `num` behandeln, jedoch weist er die gleichen Eigenschaften und die gleiche Funktion wie `num` auf.

Beschreibung

Der Datentyp `dnum` kann folgende Werte haben:

- Eine Ganzzahl, z. B. -5
- Eine Dezimalzahl, z. B. 3,45

Er lässt sich auch in Exponentialschreibweise ausdrücken, z. B. 2E3 (= 2*10^3 = 2000), 2.5E-2 (= 0.025).

Ganzzahlen zwischen -4503599627370496 und +4503599627370496 werden immer als exakte Ganzzahlen gespeichert.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `dnum`:

Beispiel 1:

```
VAR dnum reg1;
...
reg1:=1000000;
```

`reg1` wird der Wert `1000000` zugewiesen.

Beispiel 2:

```
VAR dnum hex;
Var dnum bin;
VAR dnum oct;
! Hexadecimal representation of decimal value 4294967295
hex := 0xFFFFFFFF;
! Binary representation of decimal value 255
bin := 0b11111111;
! Octal representation of decimal value 255
oct := 0o377;
```

Beispiel 3

```
VAR dnum a:=0;
VAR dnum b:=0;
a := 10 DIV 3;
b := 10 MOD 3;
```

Ganzzahldivision, wobei `a` eine Ganzzahl (=3) und `b` der Rest (=1) zugewiesen wird.

Einschränkungen

Literalwerte zwischen -4503599627370496 und 4503599627370496, die einer `dnum` zugewiesen sind, werden als exakte Ganzzahlen gespeichert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Wenn ein als `num` interpretierter literaler Wert als `dnum` zugeordnet/verwendet wird, wird er automatisch zu einem `dnum`.

Weitere Informationen

Information über	Siehe
Numerische Werte nutzen den Datentyp `num`	*num - Numerische Werte auf Seite 1740*
Numerische Ausdrücke	*Technisches Referenzhandbuch - RAPID Überblick*, Abschnitt *Grundlagen der Programmierung mit RAPID*
Operationen mit numerischen Werten	*Technisches Referenzhandbuch - RAPID Überblick*, Abschnitt *Grundlagen der Programmierung mit RAPID*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.26 errdomain – Fehlerdomäne

Verwendung

`errdomain` (*error domain*) wird verwendet, um eine Fehlerdomäne anzugeben.

Beschreibung

Daten des Typs `errdomain` stellen die Domäne dar, in der der Fehler, die Warnung oder die Statusänderung protokolliert wurde.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `errdomain`:

Beispiel 1:

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Wenn ein Fehler in der Interrupt-Routine `trap_err` festgehalten wird, werden Fehlerdomäne, Fehlernummer und Fehlertyp in geeigneten Variablen gespeichert.

Vordefinierte Daten

Die folgenden vordefinierten Konstanten können für die Angabe einer Fehlerdomäne benutzt werden.

Name	Fehlerdomäne	Wert
COMMON_ERR	Alle Fehler- und Statusänderungsdomänen	0
OP_STATE	Änderung des Betriebsstatus	1
SYSTEM_ERR	Systemfehler	2
HARDWARE_ERR	Hardwarefehler	3
PROGRAM_ERR	Programmfehler	4
MOTION_ERR	Bewegungsfehler	5
OPERATOR_ERR	Bedienungsfehler - veraltet, nicht mehr verwendet	6
IO_COM_ERR	E/A- und Kommunikationsfehler	7
USER_DEF_ERR	Anwenderdefinierte Fehler (durch RAPID erzeugt)	8
SAFETY_ERR	Sicherheitsereignisse	9
PROCESS_ERR	Prozessfehler	11
CFG_ERR	Konfigurationsfehler	12

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Eigenschaften

`errdomain` ist ein Alias-Datentyp für "num" und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Anfordern eines Interrupts bei Fehlern	*IError - Anfordern eines Interrupts bei Fehlern auf Seite 255*
Fehlernummern	*Bedienungsanleitung - Fehlerbehebung an IRC5*
Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.27 errnum – Fehlernummer

Verwendung

`errnum` wird verwendet, um alle behebbaren (nicht schwerwiegenden) Fehler zu beschreiben, die bei der Programmabarbeitung auftreten können, z. B. Division durch null.

Beschreibung

Wenn der Roboter bei der Programmabarbeitung auf einen Fehler trifft, kann dies mit der Fehlerbehandlung der Routine behandelt werden. Beispiele für solche Fehler sind Werte, die zu hoch sind, oder Division durch Null. Der Systemvariablen `ERRNO` vom Typ `errnum` werden daher abhängig von der Art eines Fehlers verschiedene Werte zugewiesen Die Fehlerbehandlungsroutine kann eventuell einen Fehler korrigieren, indem sie diese Variable liest, und anschließend kann die Programmabarbeitung fortgesetzt werden.

Ein Fehler kann auch mithilfe der Instruktion `RAISE` vom Programm aus erzeugt werden. Diesen speziellen Fehler kann die Fehlerbehandlungsroutine erkennen, wenn Sie eine Fehlernummer (im Bereich von 1-90 oder mit der Instruktion `BookErrNo` vergeben) als Argument zu `RAISE` angeben.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `errnum`:

Beispiel 1:

```
reg1 := reg2 / reg3;
...
ERROR
  IF ERRNO = ERR_DIVZERO THEN
    reg3 := 1;
    RETRY;
  ENDIF
```

Wenn `reg3 = 0` ist, erkennt der Roboter einen Fehler, sobald die Division stattfindet. Dieser Fehler kann erkannt und korrigiert werden, indem Sie `reg3` den Wert `1` zuweisen. Anschließend kann die Division erneut erfolgen und die Programmabarbeitung kann fortfahren.

Beispiel 2:

```
CONST errnum machine_error := 1;
...
IF di1=0 RAISE machine_error;
...
ERROR
  IF ERRNO=machine_error RAISE;
```

In einer Maschine tritt ein Fehler auf (erkannt durch das Eingangssignal `di1`). Ein Sprung erfolgt in die Fehlerbehandlung in der Routine, die wiederum die Fehlerbehandlung der aufrufenden Routine aufruft, in der möglicherweise der Fehler korrigiert werden kann. Mit der Konstanten `machine_error` wird der Fehlerbehandlungsroutine exakt mitgeteilt, welcher Fehlertyp aufgetreten ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Vordefinierte Daten

Anhand der Systemvariablen ERRNO kann der Typ des neuesten aufgetretenen Fehlers gelesen werden. Eine Reihe vordefinierter Konstanten kann benutzt werden, um den Typ des aufgetretenen Fehlers zu bestimmen.

Name	Fehlerursache
ERR_ACC_TOO_LOW	Zu geringe Beschleunigung/Verzögerung in der Instruktion PathAccLim oder WorldAccLim.
ERR_ACTIV_PROF	Fehler in aktiviertem Profil.
ERR_ALIASCAM_DEF	Die Kamera in Argument CameraName oder der/die/das cameradev in Argument FromCamera ist nicht in der Konfiguration des Systemparameters Communication definiert. Oder der/die/das ToCamera ist nicht im RAPID-Programm deklariert oder bereits in der Konfiguration des Systemparameters Communication definiert.
ERR_ALIASIO_DEF	FromSignal ist nicht in der E/A-Konfiguration definiert oder ToSignal ist nicht im RAPID-Programm deklariert oder es ist in der E/A-Konfiguration definiert. Instruktion AliasIO.
ERR_ALIASIO_TYPE	Die Signaltypen für die Argumente FromSignal und ToSignal sind nicht identisch (signalx). Instruktion AliasIO.
ERR_ALRDYCNT	Die Interrupt-Variable ist bereits mit einer TRAP-Routine verbunden.
ERR_ALRDY_MOVING	Der Roboter bewegt sich bereits bei der Abarbeitung der Instruktion StartMove oder StartMoveRetry.
ERR_AO_LIM	Der Wert des analogen Signals liegt außerhalb der zulässigen Grenzen
ERR_ARGDUPCND	Mehrere bedingte Argumente sind für denselben Parameter vorhanden.
ERR_ARGNAME	Argument ist ein Ausdruck, nicht vorhanden oder ist beim Abarbeiten von ArgName vom Typ switch.
ERR_ARGNOTPER	Argument ist kein Verweis auf eine Persistente.
ERR_ARGNOTVAR	Argument ist kein Verweis auf eine Variable.
ERR_ARGVALERR	Argumentwertfehler.
ERR_AXIS_ACT	Achse ist nicht aktiv.
ERR_AXIS_IND	Achse ist nicht unabhängig.
ERR_AXIS_MOVING	Achsenbewegung.
ERR_AXIS_PAR	Parameter "Achse"in der Instruktion oder in der Funktion ist falsch.
ERR_BUSSTATE	Es wird IOEnable ausgeführt, und das E/A-Netzwerk weist einen Fehlerzustand auf oder geht in den Fehlerzustand über, bevor das E/A-Gerät aktiviert wird
ERR_BWDLIMIT	Grenze StepBwdPath.
ERR_CALC_NEG	StrDig negativer Berechnungsfehler.
ERR_CALC_OVERFLOW	StrDig Berechnung Überlauf.
ERR_CALC_DIVZERO	StrDig-Division durch null.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.27 errnum – Fehlernummer
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_CALLPROC	Prozeduraufruffehler (keine Prozedur) zur Laufzeit (späte Bindung).
ERR_CAM_BUSY	Die Kamera ist mit einer anderen Anforderung belegt und kann die aktuelle Anweisung nicht durchführen.
ERR_CAM_COM_TIMEOUT	Bei der Kommunikation mit der Kamera ist ein Timeout aufgetreten. Die Kamera reagiert nicht.
ERR_CAM_GET_MISMATCH	Der mit der Instruktion CamGetParameter von der Kamera abgerufene Parameter hat den falschen Datentyp.
ERR_CAM_MAXTIME	Timeout beim Abarbeiten einer CamLoadJob- oder CamGetResult-Instruktion.
ERR_CAM_NO_MORE_DATA	Es können keine weiteren Vision-Ergebnisse abgerufen werden.
ERR_CAM_NO_PROGMODE	Die Kamera befindet sich nicht im Programmmodus.
ERR_CAM_NO_RUNMODE	Die Kamera befindet sich nicht im Abarbeitungsmodus.
ERR_CAM_NOT_ON_NETWORK	Die Kamera ist nicht verbunden.
ERR_CAM_SET_MISMATCH	Der auf die Kamera mit der Instruktion CamSetParameter geschriebene Parameter hat den falschen Datentyp oder der Wert liegt außerhalb des zulässigen Bereichs.
ERR_CFG_INTERNAL	Interner Parameter darf nicht gelesen oder beschrieben werden.
ERR_CFG_ILL_DOMAIN	Die cfgdomain, die in der Instruktion SaveCfgData verwendet wurde, ist ungültig oder wird nicht verwendet.
ERR_CFG_ILLTYPE	Nicht passender Typ - ReadCfgData, WriteCfgData.
ERR_CFG_LIMIT	Datengrenze - WriteCfgData.
ERR_CFG_NOTFND	Nicht gefunden - ReadCfgData, WriteCfgData.
ERR_CFG_OUTOFBOUNDS	Wenn ListNo bei der Eingabe -1 oder größer ist, die Anzahl der verfügbaren Instanzen - ReadCfgData, WriteCfgData.
ERR_CFG_WRITEFILE	Das Verzeichnis existiert nicht, oder der verwendete FilePath und die verwendete File sind ein Verzeichnis, oder es gibt ein anderes Problem bei der Verwendung der Instruktion SaveCfgData.
ERR_CNTNOTVAR	CONNECT-Ziel ist kein Verweis auf eine Variable.
ERR_CNV_NOT_ACT	Der Conveyor ist nicht aktiviert.
ERR_CNV_CONNECT	Die Instruktion WaitWobj ist bereits aktiv.
ERR_CNV_DROPPED	Das von der Instruktion WaitWObj erwartete Objekt wurde gelöscht.
ERR_CNV_OBJ_LOST	Das Objekt, auf das die Instruktion WaitWObj oder WaitSensor gewartet hat, hat das StartwindowWidth überschritten, ohne dass eine Verbindung hergestellt wurde.
ERR_COLL_STOP	Unterbrechung der Bewegung aufgrund einer Bewegungskollision.
ERR_CONC_MAX	Die Anzahl der aufeinander folgenden Bewegungsinstruktionen, die das Argument \Conc verwenden, wurde überschritten.
ERR_COMM_INIT	Kommunikationsinterface konnte nicht initialisiert werden.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	Fehlerursache
ERR_DEV_MAXTIME	Timeout beim Abarbeiten einer ReadBin, ReadNum, ReadStr, ReadStrBin, ReadAnyBin, or a ReadRawBytes-Instruktion.
ERR_DIPLAG_LIM	Zu großes DipLag in der Instruktion TriggSpeed, die mit aktuellem TriggL/TriggC/TriggJ/CapL/CapC verbunden ist.
ERR_DIVZERO	Division durch null.
ERR_EXCRTYMAX	Überschreitung der maximalen Anzahl an Wiederholungen. RAISE oder TRYNEXT kann für die Behebung dieses Fehlers verwendet werden.
ERR_EXECPHR	Es wurde versucht, eine Instruktion mit einem Platzhalter abzuarbeiten.
ERR_FILEACC	Falscher Zugriff auf eine Datei.
ERR_FILEEXIST	Eine Datei existiert bereits.
ERR_FILEOPEN	Eine Datei kann nicht geöffnet werden.
ERR_FILNOTFND	Datei nicht gefunden.
ERR_FNCNORET	Kein Rückgabewert.
ERR_FRAME	Neues Koordinatensystem kann nicht berechnet werden.
ERR_GO_LIM	Der Wert des analogen Gruppensignals ist außerhalb des gültigen Bereichs.
ERR_ILLDIM	Falsche Dimension des Datenfelds.
ERR_ILLQUAT	Versuch, unzulässigen Wert für Orientierung (Quaternion) zu verwenden.
ERR_ILLRAISE	Fehlernummer in RAISE außerhalb des zulässigen Bereichs.
ERR_INDCNV_ORDER	Vor der Ausführung einer Instruktion muss IndCnvInit ausgeführt werden.
ERR_INOISSAFE	Wenn versucht wird, einen sicheren Interrupt mit ISleep vorübergehend zu deaktivieren.
ERR_INOMAX	Keine Interruptnummern mehr verfügbar.
ERR_INT_NOTVAL	Keine gültige Ganzzahl, Dezimalwert.
ERR_INT_MAXVAL	Keine gültige Ganzzahl, Wert zu groß oder zu klein.
ERR_INVDIM	Dimensionen sind ungleich.
ERR_IODISABLE	Timeout beim Abarbeiten von IODisable.
ERR_IOENABLE	Timeout beim Abarbeiten von IOEnable.
ERR_IOERROR	E/A-Fehler von Instruktion Save, Load und WaitLoad.
ERR_LINKREF	Verweisfehler in der Programmtask.
ERR_LOADED	Das Programmmodul ist bereits geladen.
ERR_LOADID_FATAL	Nur interne Verwendung in LoadId und ManLoadIdProc.
ERR_LOADID_RETRY	Nur interne Verwendung in LoadId.
ERR_LOADNO_INUSE	Der Ladevorgang wird bereits in StartLoad verwendet.
ERR_LOADNO_NOUSE	Der Ladevorgang wird nicht in CancelLoad verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.27 errnum – Fehlernummer
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_MODULE	Falscher Modulname in Instruktion Save und EraseModule.
ERR_MOD_NOT_LOADED	Das Modul ist nicht vorhanden, das Symbol ist kein Modul oder der Name war für ein Symbol zu lang. Fehler von Funktion ModTimeDnum.
ERR_NAME_INVALID	Der Name des E/A-Geräts existiert nicht.
ERR_NO_ALIASIO_DEF	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion AliasIO definiert wurde.
ERR_NORUNUNIT	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.
ERR_NO_SGUN	Der angegebene Servowerkzeugname bezeichnet kein konfiguriertes Servowerkzeug.
ERR_NOTARR	Daten sind kein Datenfeld.
ERR_NOTEQDIM	Die Datenfelddimension, die beim Aufruf der Routine benutzt wurde, passt nicht zu ihren Parametern.
ERR_NOTINTVAL	Kein ganzzahliger Wert.
ERR_NOTPRES	Ein Parameter wird verwendet, obwohl das entsprechende Argument nicht beim Routineaufruf benutzt wurde.
ERR_NOTSAVED	Modul wurde nach dem Laden ins System geändert.
ERR_NOT_MOVETASK	Angegebene Task ist eine Nicht-Bewegungstask.
ERR_NUM_LIMIT	Der Wert liegt über 3.40282347E+38 oder unter -3.40282347E+38.
ERR_ORIENT_VALUE	Falscher Ausrichtungswert in der NOrient-Funktion.
ERR_OUTOFBND	Der Datenfeldindex liegt außerhalb der zulässigen Grenzen.
ERR_OVERFLOW	Überlauf der Uhr.
ERR_OUTSIDE_REACH	Die Position (robtarget) befindet sich außerhalb des Arbeitsbereichs des Roboters für die Funktion CalcJoinT.
ERR_PATH	Fehlende Zielbahn in Instruktion Save.
ERR_PATHDIST	Zu große Wiederanfahrstrecke für Instruktion StartMove, StartMoveRetry oder SetLeadThrough.
ERR_PATH_STOP	Unterbrechung der Bewegung aufgrund eines Prozessfehlers.
ERR_PERSSUPSEARCH	Die persistente Variable ist bereits zu Beginn des Suchprozesses TRUE.
ERR_PID_MOVESTOP	Nur interne Verwendung in LoadId und ManLoadIdProc.
ERR_PID_RAISE_PP	Fehler von ParIdRobValid, ParIdPosValid, LoadId oder ManLoadIdProc.
ERR_PRGMEMFULL	Programmspeicher ist voll.
ERR_PROCSIGNAL_OFF	Prozess-Signal ist aus.
ERR_PROGSTOP	Der Roboter befindet sich bei Ausführung der Instruktion StartMove, StartMoveRetry oder SetLeadThrough im Zustand Programmstopp.
ERR_RANYBIN_CHK	Prüfsummenfehler bei Datenübertragung mit der Instruktion ReadAnyBin.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	Fehlerursache
ERR_RANYBIN_EOF	Dateiende erkannt, bevor alle Bytes in der Instruktion ReadAnyBin oder ReadRawBytes gelesen wurden.
ERR_RCVDATA	Es wurde versucht, mit ReadNum nicht numerische Daten zu lesen.
ERR_REFUNKDAT	Verweis auf vollkommen unbekanntes Datenobjekt.
ERR_REFUNKFUN	Verweis auf unbekannte Funktion.
ERR_REFUNKPRC	Verweis auf unbekannte Prozedur beim Verbinden oder während der Laufzeit (späte Bindung).
ERR_REFUNKTRP	Verweis auf unbekannten Interrupt.
ERR_RMQ_DIM	Falsche Dimensionen. Die Dimensionen der angegebenen Daten stimmen nicht mit den Dimensionen der Daten in der Meldung überein.
ERR_RMQ_FULL	Die Meldungswarteschlange des Ziels ist voll.
ERR_RMQ_INVALID	Der Ziel-Slot ist nicht vorhanden oder ungültig.
ERR_RMQ_INVMSG	Ungültige Meldung, wahrscheinlich von anderem Client als einem RAPID-Task gesendet.
ERR_RMQ_MSGSIZE	Die Meldung ist zu groß. Verringern Sie die Meldungsgröße.
ERR_RMQ_NAME	Der angegebene Slot-Name ist ungültig oder wurde nicht gefunden.
ERR_RMQ_NOMSG	Keine Meldung in der Warteschlange, wahrscheinlich aufgrund eines Stromausfalls.
ERR_RMQ_TIMEOUT	Timeout während des Wartens auf eine Antwort in RMQSendWait oder RMQReadWait.
ERR_RMQ_VALUE	Die Wertsyntax stimmt nicht mit dem Datentyp überein.
ERR_ROBLIMIT	Die Position ist erreichbar, aber mindestens eine Achse liegt außerhalb der Achsenbeschränkungen oder die Grenzen für mindestens eine verbundene Achse werden überschritten (Funktion CalcJoinT).
ERR_SC_WRITE	Fehler beim Senden an einen externen Computer.
ERR_SGUN_ESTOP	Not-Halt bei der Servowerkzeugbewegung.
ERR_SGUN_MOTOFF	Die Instruktion wird von einem Hintergrundtask aufgerufen und das System befindet sich im Zustand „Motoren aus“.
ERR_SGUN_NEGVAL	Das Argument PrePos ist mit einem Wert angegeben, der kleiner als Null ist.
ERR_SGUN_NOTACT	Die mechanische Einheit des Servowerkzeugs ist nicht aktiviert.
ERR_SGUN_NOTINIT	Die Position des Servowerkzeugs wurde nicht initialisiert.
ERR_SGUN_NOTOPEN	Die Zange ist nicht geöffnet, wenn die Instruktion aufgerufen wird.
ERR_SGUN_NOTSYNC	Die Elektroden des Servowerkzeugs sind nicht synchronisiert.
ERR_SIGSUPSEARCH	Das Signal hat bereits am Beginn des Suchvorgangs einen positiven Wert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.27 errnum – Fehlernummer
RobotWare - OS
Fortsetzung

Name	Fehlerursache
ERR_SIG_NOT_VALID	Auf das E/A-Signal kann nicht zugegriffen werden, weil möglicherweise das E/A-Gerät nicht in Betrieb ist oder weil ein Fehler in der Konfiguration vorliegt (gilt nur für ICI-Feldbus).
ERR_SOCK_ADDR_INVALID	Die angegebene Adresse ist ungültig.
ERR_SOCK_ADDR_INUSE	Die Adresse und der Port werden bereits verwendet und können nicht erneut verwendet werden. Verwenden Sie in SocketBind eine andere Portnummer oder Adresse.
ERR_SOCK_CLOSED	Der Socket ist geschlossen oder wurde nicht erstellt.
ERR_SOCK_IS_BOUND	Der Socket ist bereits an eine Adresse gebunden und kann nicht erneut gebunden werden.
ERR_SOCK_IS_CONN	Der Socket ist verbunden.
ERR_SOCK_NET_UNREACH	Das Netzwerk ist nicht erreichbar oder die Verbindung geht nach dem Öffnen eines Sockets verloren.
ERR_SOCK_NOT_BOUND	Der Socket wurde nicht an eine Adresse gebunden.
ERR_SOCK_NOT_CONN	Der Socket ist nicht verbunden.
ERR_SOCK_TIMEOUT	Innerhalb der Frist des Timeouts konnte keine Verbindung hergestellt oder keine Daten empfangen werden.
ERR_SOCK_UNSPEC	Nicht spezifizierte Ausnahme vom zugrundeliegenden Aufruf und das Betriebssystem.
ERR_SPEED_REFRESH_LIM	Override außerhalb des zulässigen Bereichs in SpeedRefresh.
ERR_SPEEDLIM_VALUE	Die in den Instruktionen SpeedLimAxis und SpeedLimCheckPoint verwendete Geschwindigkeit ist zu niedrig.
ERR_STARTMOVE	Der Roboter befindet sich bei Ausführung der Instruktion StartMove, StartMoveRetry oder SetLeadThrough im Haltezustand.
ERR_STORE_PROF	Fehler in gespeichertem Profil.
ERR_STRTOOLNG	Die Zeichenfolge ist zu lang.
ERR_SYM_ACCESS	Zugriffsfehler beim Lesen/Schreiben eines Symbols.
ERR_SYMBOL_TYPE	Das Datenobjekt und die im ArgumentValue verwendete Variable weisen unterschiedliche Typen auf. Wenn ALIAS-Datentypen verwendet werden, erhalten Sie ebenfalls diesen FEHLER, auch wenn die Typen den gleichen Basisdatentyp aufweisen. Instruktionen GetDataVal, SetDataVal und SetAllDataVal.
ERR_SYNCMOVEOFF	Timeout von SyncMoveOff.
ERR_SYNCMOVEON	Timeout von SyncMoveOn.
ERR_SYNTAX	Syntaxfehler im geladenen Modul.
ERR_TASKNAME	Taskname nicht im System gefunden.
ERR_TP_DIBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Eingang unterbrochen.
ERR_TP_DOBREAK	Eine Read-Instruktion vom FlexPendant wurde durch einen digitalen Ausgang unterbrochen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	Fehlerursache
ERR_TP_MAXTIME	Timeout beim Abarbeiten einer read-Instruktion vom FlexPendant.
ERR_TP_NO_CLIENT	Es ist kein Client für Interaktionen vorhanden bei Verwendung einer Instruktion vom FlexPendant.
ERR_TRUSTLEVEL	Das E/A-Gerät darf nicht deaktiviert werden.
ERR_TXTNOEXIST	Falsche Tabelle oder Index in der Funktion TextGet.
ERR_UDPUC_COMM	Zeitüberschreitung der Kommunikation für das UdpUc-Gerät.
ERR_UI_INITVALUE	Es liegt ein Anfangswertfehler in der Funktion UINumEntry vor.
ERR_UI_MAXMIN	Der Mindestwert in der Funktion UINumEntry, UIDnumEntry, UINumTune oder UIDnumTune ist größer als der Höchstwert.
ERR_UI_NOTINT	Der Wert ist keine Ganzzahl, wenn festgelegt wurde, dass bei Verwendung von UINumEntry oder UIDnumEntry eine Ganzzahl verwendet werden soll.
ERR_UISHOW_FATAL	Anderer Fehler als ERR_TP_NO_CLIENT oder ERR_UISHOW_FULL in Instruktion UIShow.
ERR_UISHOW_FULL	Bei Verwendung der Instruktion UIShow ist auf dem FlexPendant kein Arbeitsspeicher für eine weitere Anwendung vorhanden.
ERR_UNIT_PAR	Parameter Mech_unit in TestSignDefine ist falsch.
ERR_UNKINO	Unbekannte Interrupt-Nummer bei Ausführung der Instruktionen IWatch oder ISleep.
ERR_UNKPROC	Fehlerhafter Verweis auf Ladevorgang in Instruktion WaitLoad.
ERR_UNLOAD	Entladefehler in Instruktion UnLoad oder WaitLoad.
ERR_USE_PROF	Fehler in verwendetem Profil.
ERR_WAITSYNCTASK	Timeout von WaitSyncTask.
ERR_WAIT_MAXTIME	Timeout beim Abarbeiten einer WaitDI, WaitDO, WaitAI, WaitAO, WaitGI, WaitGO, WaitUntil, WaitSensor oder WaitWObj-Instruktion.
ERR_WHLSEARCH	Kein Suchstopp.
ERR_WOBJ_MOVING	Die mechanische Einheit mit Arbeitsobjekt bewegt CalcJointT.

Eigenschaften

errnum ist ein Alias-Datentyp für num und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Fehlerbehandlung	*Technisches Referenzhandbuch - RAPID Overview*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.28 errstr – Fehlerreihe

Verwendung

`errstr` wird zum Schreiben in Fehlermeldungen verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `errstr`:

Beispiel 1:

```
VAR errstr arg:= "This is an example";

ErrLog 5100, \W, ERRSTR_TASK, ERRSTR_CONTEXT, arg, ERRSTR_EMPTY,
     ERRSTR_UNUSED;
```

Vordefinierte Daten

Name	Beschreibung
ERRSTR_EMPTY	Argument ist leer.
ERRSTR_UNUSED	Argument wird nicht verwendet.
ERRSTR_TASK	Name der aktuellen Task
ERRSTR_CONTEXT	Kontext

Eigenschaften

`errstr` ist ein Alias-Datentyp für `string` und übernimmt daher seine Eigenschaften.

Weitere Informationen

Information über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data Types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.29 errtype – Fehlertyp

Verwendung

`errtype` (*error type*) wird verwendet, um einen Fehlertyp anzugeben.

Beschreibung

Daten des Typs `errtype` stellen den Typ (Statusänderung, Warnung, Fehler) einer Fehlermeldung dar.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `errtype`:

Beispiel 1:

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Wenn ein Fehler in der Interrupt-Routine `trap_err` festgehalten wird, werden Fehlerdomäne, Fehlernummer und Fehlertyp in geeigneten Variablen gespeichert.

Vordefinierte Daten

Die folgenden vordefinierten Konstanten können für die Angabe eines Fehlertyps benutzt werden.

Name	Fehlertyp	Wert
TYPE_ALL	Beliebiger Fehlertyp (Statusänderungen, Warnungen, Fehler)	0
TYPE_STATE	Statusänderung (Betriebsmeldung)	1
TYPE_WARN	Warnung (z. B. durch RAPID behebbarer Fehler)	2
TYPE_ERR	Fehler	3

Eigenschaften

`errtype` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Anfordern eines Interrupts bei Fehlern	*IError - Anfordern eines Interrupts bei Fehlern auf Seite 255*
Fehlernummern	*Bedienungsanleitung - Fehlerbehebung an IRC5*
Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.30 event_type – Typ der Ereignisroutine

Verwendung

`event_type` stellt den aktuellen Ereignisroutinentyp mit einer symbolischen Konstanten dar.

Beschreibung

Mit der Funktion `EventType` lässt sich prüfen, ob der aktuelle RAPID-Code aufgrund eines bestimmten Systemereignisses ausgeführt wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `event_type`:

Beispiel 1:

```
VAR event_type my_type;
...
my_type := EventType( );
```

Der Ereignisroutinentyp, der abgearbeitet wird, wird in der Variablen `my_type` gespeichert.

Vordefinierte Daten

Folgende Konstanten des Typs `event_type` sind vordefiniert:

RAPID Konstante	Wert	Abgearbeiteter Ereignistyp
`EVENT_NONE`	0	Kein Ereignis wird abgearbeitet.
`EVENT_POWERON`	1	POWER_ON-Ereignis
`EVENT_START`	2	START-Ereignis
`EVENT_STOP`	3	STOP-Ereignis
`EVENT_QSTOP`	4	QSTOP-Ereignis
`EVENT_RESTART`	5	RESTART-Ereignis
`EVENT_RESET`	6	RESET -Ereignis
`EVENT_STEP`	7	STEP -Ereignis

Eigenschaften

`event_type` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Ereignisroutinen im Allgemeinen	*Technisches Referenzhandbuch - Systemparameter*, Abschnitt *Controller - Event Routine*
Abrufen eines Ereignistyps	*EventType – Abrufen des aktuellen Ereignistyps in einer beliebigen Ereignisroutine auf Seite 1299*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.31 exec_level – Abarbeitungsebene

Verwendung

Mit `exec_level` kann die Programmabarbeitungsebene angegeben werden.

Beschreibung

Mit der Funktion `ExecLevel` kann die aktuelle Abarbeitungsebene des gegenwärtig ausgeführten `RAPID`-Codes abgerufen werden.

Vordefinierte Daten

Die folgenden Konstanten vom Typ `exec_level` sind vordefiniert:

RAPID Konstante	Wert	Abarbeitungsebene
`LEVEL_NORMAL`	0	Abarbeitung auf Basisebene
`LEVEL_TRAP`	1	Abarbeiten in Trap-Routine
`LEVEL_SERVICE`	2	Abarbeiten in Serviceroutine [i]

[i] Mit `LEVEL_SERVICE` ist jede Routine, Serviceroutine (einschließlich Aufrufroutine) und Interrupt-Routine von Systemeingangssignalen gemeint.

Eigenschaften

`exec_level` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Ruft die aktuelle Abarbeitungsebene ab.	*ExecLevel – Abarbeitungsebene abrufen auf Seite 1302*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.32 extjoint - Position von externen Achsen

Verwendung

`extjoint` wird zur Definition der Achsenpositionen von zusätzlichen Achsen, Positionierern oder Werkstückmanipulatoren verwendet.

Beschreibung

Der Roboter kann zusätzlich zu seinen sechs internen Achsen bis zu sechs zusätzliche Achsen steuern, also insgesamt zwölf Achsen Diese sechs zusätzlichen Achsen werden logisch bezeichnet: a, b, c, d, e, f. Jede solche logische Achse kann mit einer physischen Achse verbunden werden und in diesem Fall wird die Verbindung in den Systemparametern definiert.

Daten des Typs `extjoint` werden für Positionswerte für jede der logischen Achsen (a-f) verwendet.

Für jede mit einer physischen Achse verbundene logische Achse wird die Position wie folgt definiert:

- Für Rotationsachsen - Die Position wird als Drehung in Grad ab der Kalibrierposition definiert.
- Für lineare Achsen - Die Position wird als Abstand in mm ab der Kalibrierposition definiert.

Wenn eine logische Achse nicht mit einer physischen Achse verbunden ist, wird dies mit dem Positionswert 9E9 angezeigt. Bei der Abarbeitung werden die Positionsdaten jeder Achse überprüft. Außerdem wird geprüft, ob die entsprechende Achse verbunden ist. Wenn der gespeicherte Positionswert nicht mit der aktuellen Achsenverbindung übereinstimmt, gilt Folgendes:

- Wenn die Position nicht in den Positionsdaten definiert ist (Wert ist 9E9), wird der Wert ignoriert, falls die Achse verbunden und nicht aktiviert ist. Wenn aber die Achse aktiviert ist, ergibt das einen Fehler.
- Wenn die Position in den Positionsdaten definiert ist, obwohl die Achse nicht verbunden ist, wird der Wert ignoriert.

Es wird keine Bewegung durchgeführt, aber für eine Achse mit gültigen Positionsdaten wird kein Fehler ausgegeben, wenn die Achse nicht aktiviert wird.

Wenn ein zusätzlicher Achsenversatz verwendet wird (Instruktion `EOffsOn` oder `EOffsSet`), werden die Positionen im `ExtOffs`-Koordinatensystem angegeben.

Wenn eine zusätzlich Achse im unabhängigen Modus arbeitet und vom Roboter und seinen zusätzlichen Achsen eine neue Bewegung ausgeführt werden soll, darf als Positionsdaten für die zusätzliche Achse im unabhängigen Modus nicht 9E9 verwendet werden. Die Daten müssen ein beliebiger Wert sein, der nicht vom System verwendet wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `extjoint`:

Beispiel 1:

```
VAR extjoint axpos10 := [ 11, 12.3, 9E9, 9E9, 9E9, 9E9] ;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Position eines externen Positioniergeräts, `axpos10`, wird wie folgt definiert:

- Die Position der externen logischen Achse „a“ ist auf `11` gesetzt, ausgedrückt in Grad oder mm (abhängig vom Typ der Achse).
- Die Position der externen logischen Achse „b“ ist auf `12.3` gesetzt, ausgedrückt in Grad oder mm (abhängig vom Typ der Achse).
- Achsen c bis f sind undefiniert.

Komponenten

`eax_a`

external axis a

Datentyp: `num`

Die Position der externen logischen Achse a, ausgedrückt in Grad oder mm (abhängig vom Typ der Achse)

`...`

`eax_f`

external axis f

Datentyp: `num`

Die Position der externen logischen Achse f, ausgedrückt in Grad oder mm (abhängig vom Typ der Achse).

Struktur

```
< dataobject of extjoint >
  < eax_a of num >
  < eax_b of num >
  < eax_c of num >
  < eax_d of num >
  < eax_e of num >
  < eax_f of num >
```

Weitere Informationen

Information über	Siehe
Positionsdaten	*robtarget - Positionsdaten auf Seite 1778* *jointtarget – Achsenpositionsdaten auf Seite 1720*
ExtOffs-Koordinatensystem	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.33 flypointdata - Daten für Flying Start/End

Verwendung

`flypointdata` werden verwendet, um alle Daten von Flying Start oder Flying End eines CAP-Prozesses zu definieren. Sie gehören zu den `capdata` sowohl des Flying Start als auch des Flying End.

Definitionen

`flypointdata` definiert die Daten für den fliegenden Start und das fliegende Ende:

- Diese Funktionalität steht nur für CAP zur Verfügung.
- Flying Start wird durch die Kombination von *first instruction = TRUE* und Zonenpunkt ausgelöst.
- Flying End wird durch die Kombination von *last_instr = TRUE* und Zonenpunkt ausgelöst.
- Pendelstart wird ignoriert.
- Wenn der Ausgangspunkt ein Feinpunkt ist, wird der Flying Start nicht ausgeführt.
- Wenn der Endpunkt ein Feinpunkt ist, wird Flying End nicht ausgeführt.
- Bewegungsverzögerung wird ignoriert.
- Ein Neustart nach einem Fehler wird in der gleichen Weise ausgeführt wie gewohnt. Es gibt keine Besonderheiten für Flying Start, Scrape-Start steht zur Verfügung, wenn der Anwendungsprozess aktiv war, als der Fehler auftrat.
- Ist Pendelbewegung aktiviert, erfolgt der Übergang in die Zone durch Hochfahren der Pendelmuster am Eingang der Zone, bis das vollständige Muster erreicht ist, wenn der TCP die Zone verlässt.
- Die Überwachung ist während der START-PHASE (mit Verschieben des TCP), der MAIN-Phase und der END_MAIN-Phase (mit Verschieben des TCP) aktiviert.
- Die Rückwärtsbewegung auf die Bahn wird auf das Zurückgehen auf Position 4 beschränkt (siehe folgende Abbildung).
- Der Benutzer muss Abstand, Annäherung und den Winkel des Anwendungsprozesses anpassen: zum Beispiel muss der Punkt für das Lichtbogenschweißen (Punkt 4 in der Abbildung) so platziert werden, dass gezündet werden kann.
- Der Abstand zwischen Position 4 und 6 darf nicht = 0 sein.
- Die `START process_dist` muss gleich oder kürzer sein als die `START distance`.
- Wenn die Programmabarbeitung angehalten wird und der Anwendungsprozess aktiv ist (zwischen den Positionen 3 und 6), wird sich CAP wie gewohnt verhalten, das heißt, Rückwärtsbewegung auf die Bahn (nur wenn Pos. 4 passiert wurde), Start der Pendelbewegung,

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Bewegungsverzögerung und Zeitüberschreitung des Starts der Bewegung stehen zur Verfügung.

- Wenn die Programmabarbeitung zwischen den Positionen 1 und 3 oder zwischen den Positionen 7 und 10 angehalten wird, verhält sich die `CapX`-Instruktion wie eine `TrigX`-Instruktion.
- Es empfiehlt sich, dass das erste CAP-Segment bei Flying Start mindestens `START distance` lang sein muss.
- Wenn das erste Segment kürzer ist als `START distance`, jedoch länger als `START process_dist`, verschieben sich die Positionen 2 und 4 in Richtung Position 1.
- Wenn das erste Segment kürzer ist als oder gleich dem Wert von `START process_dist`, fallen die Positionen 1 und 2 zusammen und Position 4 befindet sich am Ende des Segments.
- Es empfiehlt sich, dass das letzte CAP-Segment bei Flying End mindestens `END distance + END process_dist` lang ist.
- Wenn das letzte Segment kürzer ist als `END distance + END process_dist`, jedoch länger als `END process_dist`, verschieben sich die Positionen 7 und 9 in Richtung Position 10.
- Wenn das letzte Segment kürzer ist als oder gleich dem Wert von `END process_dist`, fallen die Positionen 8 und 10 zusammen und die Position 6 befindet sich am Anfang des Segments.
- Die Zeitüberschreitung der START-Phase, die in `capdata` angegeben ist, wird nur beim Neustart des Anwendungsprozesses verwendet.
- Wenn ein Prozessfehler auftritt, nachdem die Prefetch-Anforderung von Bewegung an der letzten CAP-Instruktion angekommen ist (nach Position 9), das heißt, PGM wurde von der CAP-Instruktion freigegeben und kann mit der nächsten Instruktion fortgesetzt werden, wird eine Fehlerprotokollnachricht gesendet und der Prozess wird gestoppt, *aber* der Roboter bewegt sich weiter.

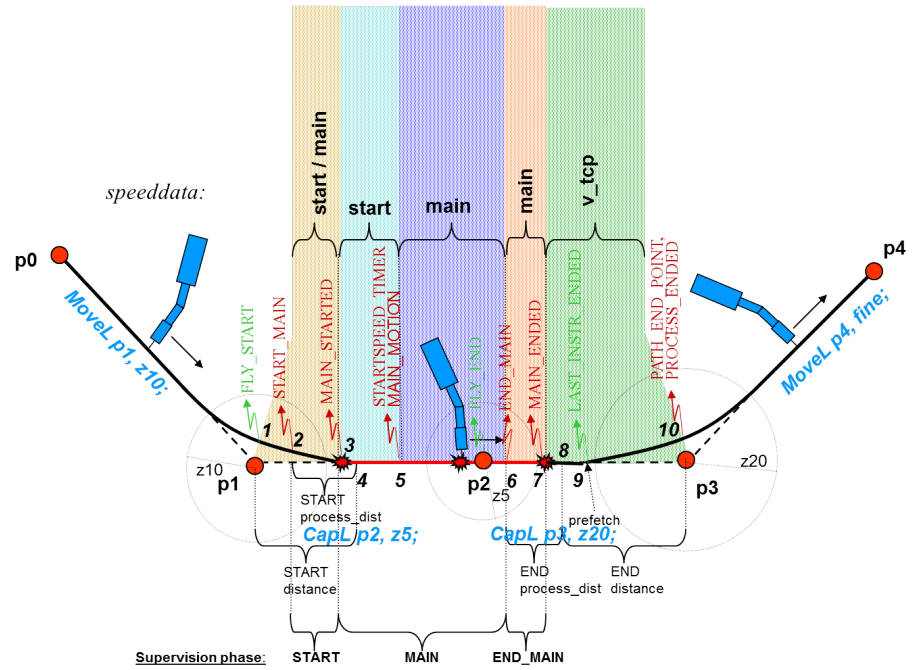
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.33 flypointdata - Daten für Flying Start/End

Continuous Application Platform (CAP)

Fortsetzung



xx1200000180

Komponenten

from_start

Datentyp: `bool`

Nicht verwendet

process_dist

Datentyp: `num`

Der Abstand (in mm), in dem der Prozess gestartet (bei *flying start)* oder beendet (bei *flying end)* wird.

distance

Datentyp: `num`

Legt Start/Ende der Überwachung des CAP-Prozesses als Abstand (in mm) vom Start-/Endpunkt fest.

Struktur

```
< databases of flypointdata >
  < from_start of bool >
  < process_dist of num >
  < distance of num >
```

Weitere Informationen

	Beschrieben in:
`capdata` Datentyp	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.34 handler_type – Typ des Abarbeitungshandlers

Verwendung

Mit `handler_type` wird der Typ des Abarbeitungshandlers einer `RAPID`-Programmroutine angegeben.

Beschreibung

Mit der Funktion `ExecHandler` lässt sich überprüfen, ob der aktuelle `RAPID`-Code in einer Abarbeitungsbehandlung in der `RAPID`-Programmroutine ausgeführt wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `handler_type`:

Beispiel 1:

```
VAR handler_type my_type;
...
my_type := ExecHandler( );
```

Der Typ der Abarbeitungsbehandlung, in der der Code ausgeführt wird. Wird in der Variablen `my_type` gespeichert.

Vordefinierte Daten

Folgende Konstanten des Typs `handler_type` sind vordefiniert:

RAPID Konstante	Wert	Typ der Abarbeitungsbehandlung
`HANDLER_NONE`	0	Wird in keiner Behandlung ausgeführt.
`HANDLER_BWD`	1	Wird im `BACKWARD`-Handler ausgeführt.
`HANDLER_ERR`	2	Wird im `ERROR`-Handler ausgeführt.
`HANDLER_UNDO`	3	Wird im `UNDO`-Handler ausgeführt.

Eigenschaften

`handler_type` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Ruft den Typ der Abarbeitungsbehandlung ab.	*ExecHandler - Abrufen des Typs der Abarbeitungsbehandlung auf Seite 1301*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.35 icondata – Daten für Icon-Anzeige

Verwendung

`icondata` wird für die Darstellung von Standard-Icons auf dem Benutzergerät wie beispielsweise dem FlexPendant verwendet.

Beschreibung

Eine Nummerierungskonstante `icondata` kann an das Argument `Icon` in den Anweisungen `UIMsgBox`,`UIMsgWrite`, `WaitAI`, `WaitAO`, `WaitDI`, `WaitDO`, `WaitGI`, `WaitGO` und `WaitUntil` sowie an die Funktionen `UIMessageBox`, `UINumEntry`, `UIDnumEntry`, `UINumTune`, `UIDnumTune`, `UIAlphaEntry` und `UIListView` übergeben.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `icondata`:

Beispiel 1:

```
VAR btnres answer;

UIMsgBox "More ?" \Buttons:=btnYesNo \Icon:=iconInfo \Result:=
     answer;
IF answer= resYes THEN
...
ELSEIF answer =ResNo THEN
...
ENDIF
```

Die standardmäßige Aufzählungskonstante `iconInfo` führt dazu, dass oben im Meldungsfenster auf der Benutzeroberfläche ein Informations-Icon angezeigt wird.

Vordefinierte Daten

Folgende Konstanten des Datentyps `icondata` sind im System vordefiniert:

Wert	Konstante	Symbol
0	`iconNone`	Kein Icon
1	`iconInfo`	Informations-Icon
2	`iconWarning`	Warn-Icon
3	`iconError`	Fehler-Icon
4	`iconQuestion`	Fragen-Symbol

Eigenschaften

`icondata` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Benutzerinteraktions-Meldungsfenster	*UIMsgBox - Benutzer-Meldungsfenster, grundlegender Typ auf Seite 1011*
Benutzerinteraktions-Meldungsfenster	*UIMessageBox - Benutzer-Meldungsfenster, erweiterter Typ auf Seite 1605*
Benutzerinteraktion "Zahleneingabe"	*UINumEntry - Eintrag eines numerischen Werts durch den Benutzer auf Seite 1614*
Benutzerinteraktion "Zahlensynchronisierung"	*UINumTune - Benutzer-Zahlensynchronisierung auf Seite 1621*
Benutzerinteraktion "Alpha-Eingabe"	*UIAlphaEntry - Benutzer-Alpha-Eintrag auf Seite 1571*
Benutzerinteraktion "Listenansicht"	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.36 identno - Identität für Bewegungsinstruktionen

Verwendung

`identno` (*Identity Number*) wird zur Steuerung der Synchronisierung von mindestens zwei koordinierten Synchronbewegungen verwendet.

Der Datentyp `identno` kann nur in einem *MultiMove*-System mit der Option *Coordinated Robots* und nur in als Motion Task definierten Programmtasks verwendet werden.

Beschreibung

Bewegungsinstruktionen in einem MultiMove-System müssen bei koordinierter synchronisierter Bewegung mit Parameter `\ID` vom Datentyp `identno` programmiert werden, und `\ID` ist in allen anderen Fällen nicht zulässig.

Die angegebene `\ID`-Nummer muss in allen kooperierenden Programmtasks identisch sein. Durch die ID-Nummer wird gewährleistet, dass die Bewegungen bei Abarbeitung nicht verwechselt werden.

Im koordinierten synchronisierten Modus muss die gleiche Anzahl an ausgeführten Bewegungsinstruktionen in allen Programmtasks vorhanden sein. Der optionale Parameter `\ID` vom Datentyp `identno` wird verwendet, um vor dem Start der Bewegungen zu überprüfen, ob alle betroffenen Bewegungsinstruktionen parallel ablaufen. Die `\ID`-Nummer muss in allen parallel ablaufenden Instruktionen identisch sein.

Der Benutzer muss keine Variable des Typs `identno` deklarieren, sondern kann direkt in den Instruktionen eine Nummer verwenden (siehe *Grundlegende Beispiele*).

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `identno`:

Beispiel 1:

```
PERS tasks task_list{2} := [["T_ROB1"],["T_ROB2"]];
VAR syncident sync1;
VAR syncident sync2;

PROC proc1()
  ...
  SyncMoveOn sync1, task_list;
  MoveL *\ID:=10,v100,z50,mytool;
  MoveL *\ID:=20,v100,fine,mytool;
  SyncMoveOff sync2;
  ...
ENDPROC
```

Eigenschaften

`identno` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data types*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.37 intnum - Interrupt-Identität

Verwendung

`intnum` (*interrupt numeric*) wird zur Identifizierung eines Interrupts verwendet.

Beschreibung

Wenn eine Variable des Typs `intnum` mit einer Interrupt-Routine verbunden wird, erhält sie einen bestimmten Wert zur Identifizierung des Interrupts. Diese Variable wird dann in allen Vorgängen mit dem Interrupt benutzt, z. B. beim Anfordern oder Blockieren eines Interrupts.

Mehrere Interrupt-Identitäten können mit derselben Interrupt-Routine verbunden sein. Anhand der Systemvariablen `INTNO` kann in einer Interrupt-Routine der Typ des aufgetretenen Interrupts bestimmt werden.

Eine Variable vom Typ `intnum` muss immer global im Modul deklariert werden.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `intnum`:

Beispiel 1:

```
VAR intnum feeder_error;
...
PROC main()
  CONNECT feeder_error WITH correct_feeder;
  ISignalDI di1, 1, feeder_error;
```

Ein Interrupt wird erzeugt, wenn der Eingang di1 auf 1 gesetzt wird. In diesem Fall erfolgt ein Aufruf an die Interrupt-Routine correct_feeder.

Beispiel 2:

```
VAR intnum feeder1_error;
VAR intnum feeder2_error;
...
PROC init_interrupt()
...
  CONNECT feeder1_error WITH correct_feeder;
  ISignalDI di1, 1, feeder1_error;
  CONNECT feeder2_error WITH correct_feeder;
  ISignalDI di2, 1, feeder2_error;
...
ENDPROC
...
TRAP correct_feeder
  IF INTNO=feeder1_error THEN
  ...
  ELSE
  ...
  ENDIF
...
ENDTRAP
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Ein Interrupt wird erzeugt, wenn der Eingang `di1` oder `di2` auf `1` gesetzt wird. Dann erfolgt ein Aufruf an die Interrupt-Routine `correct_feeder`. Anhand der Systemvariablen `INTNO` wird in einer Interrupt-Routine der Typ des aufgetretenen Interrupts ermittelt.

Einschränkungen

Die maximale Anzahl an gleichzeitig aktiven Variablen des Typs `intnum` (zwischen `CONNECT` und `IDelete`) ist auf 100 begrenzt. Die maximale Anzahl an gleichzeitigen Interrupts in der Warteschlange für die Abarbeitung der `TRAP`-Routine ist auf 30 begrenzt.

Eigenschaften

`Intnum` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Interrupts*
Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*
Verbinden von Interrupts	*CONNECT - Verbinden eines Interrupts mit einer Interrupt-Routine auf Seite 161*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.38 iodev – E/A-Gerät

Verwendung

`iodev` *(E/A-Gerät)* wird für E/A-Geräte und Dateien verwendet.

Beschreibung

Daten des Typs `iodev` enthalten den Namen einer Datei oder eines E/A-Geräts. Er kann durch die Instruktion `Open` mit einer physischen Einheit verbunden und dann zum Lesen und Schreiben benutzt werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `iodev`:

Beispiel 1:

```
VAR iodev file;
...
Open "HOME:/LOGDIR/INFILE.DOC", file\Read;
input := ReadNum(file);
```

Die Datei `INFILE.DOC` wird zum Lesen geöffnet. Beim Lesen aus der Datei wird `file` anstelle des Dateinamens als Verweis verwendet.

Eigenschaften

`iodev` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Kommunikation über E/A-Geräte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Communication*
Konfigurieren der E/A-Gerät	*Technisches Referenzhandbuch - Systemparameter*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.39 iounit_state - Status des E/A-Geräts

Verwendung

`iounit_state` wird zur Wiedergabe des aktuellen Status des E/A-Geräts verwendet.

Beschreibung

Eine `iounit_state`-Konstante dient der Prüfung des Rückgabewerts aus der Funktion `IOUnitState`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `iounit_state`:

Beispiel 1:

```
IF (IOUnitState ("UNIT1" \Phys) = IOUNIT_PHYS_STATE_RUNNING) THEN
  ! Possible to access some signal on the I/O unit
ELSE
  ! Read/Write some signal on the I/O unit result in error
ENDIF
```

Test wird durchgeführt, wenn das E/A-Gerät `UNIT1` bereit ist und läuft.

Vordefinierte Daten

Die vordefinierten symbolischen Konstanten des Datentyps `iounit_state` können in der Funktion `IOUnitState` gefunden werden.

Eigenschaften

`iounit_state` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Abrufen des aktuellen Status des E/A-Geräts	*IOUnitState - Abrufen des aktuellen Status des E/A-Geräts auf Seite 1363*
Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and Output Signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.40 jointtarget – Achsenpositionsdaten

Verwendung

`jointtarget` wird zur Definition der Position verwendet, in die sich der Roboter und die externen Achsen mit der Instruktion `MoveAbsJ` bewegen.

Beschreibung

`jointtarget` definiert jede einzelne Achsenposition für den Roboter und für die externen Achsen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `jointtarget`:

Beispiel 1:

```
CONST jointtarget calib_pos := [ [ 0, 0, 0, 0, 0, 0], [ 0, 9E9,
     9E9, 9E9, 9E9, 9E9] ];
```

Die normale Kalibrierungsposition für den IRB2400 wird in `calib_pos` mit dem Datentyp `jointtarget` definiert. Die normale Kalibrierungsposition 0 (Grad oder mm) wird ebenfalls für die externe logische Achse a definiert. Die externen Achsen b bis f sind undefiniert.

Komponenten

`robax`

robot axes

Datentyp: `robjoint`

Achsenpositionen für die Roboterachsen in Grad.

Die Achsenposition wird als Drehung in Grad für die jeweilige Achse (den Arm) in positiver oder negativer Richtung ab der Achsenkalibrierungsposition definiert.

`extax`

external axes

Datentyp: `extjoint`

Die Position der externen Achsen.

Die Position `für jede ein`zelne Achse (`eax_a`, `eax_b ... eax_f`) wird wie folgt definiert:

- Für Rotationsachsen wird die Position als Drehung in Grad ab der Kalibrierposition definiert.
- Für lineare Achsen wird die Position als Abstand in mm ab der Kalibrierposition definiert.

Externe Achsen `eax_a ...` sind logische Achsen. Die Beziehung zwischen der Nummer der logischen Achse und der Nummer der physischen Achse wird in den Systemparametern definiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Der Wert 9E9 wird für Achsen definiert, die nicht verbunden sind. Wenn sich die in den Positionsdaten definierten Achsen von den Achsen unterscheiden, die tatsächlich bei der Programmabarbeitung verbunden sind, gilt Folgendes:

- Wenn die Position nicht in den Positionsdaten definiert ist (Wert 9E9), wird der Wert ignoriert, falls die Achse verbunden und nicht aktiviert ist. Wenn aber die Achse aktiviert ist, ergibt das einen Fehler.
- Wenn die Position in den Positionsdaten definiert ist, obwohl die Achse nicht verbunden ist, wird der Wert ignoriert.

Es wird keine Bewegung durchgeführt, aber für eine Achse mit gültigen Positionsdaten wird kein Fehler ausgegeben, wenn die Achse nicht aktiviert wird.

Wenn eine externe Achse im unabhängigen Modus arbeitet und vom Roboter und seinen externen Achsen eine neue Bewegung ausgeführt werden soll, darf als Positionsdaten für die externe Achse im unabhängigen Modus nicht 9E9 verwendet werden, sondern es muss ein anderer Wert (nicht vom System verwendet) angegeben werden.

Struktur

```
< dataobject of jointtarget >
  < robax of robjoint >
    < rax_1 of num >
    < rax_2 of num >
    < rax_3 of num >
    < rax_4 of num >
    < rax_5 of num >
    < rax_6 of num >
  < extax of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
```

Weitere Informationen

Information über	Siehe
Bewegen zur Achsenposition	*MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition auf Seite 382* *MoveExtJ - Bewegen einer oder mehrerer mechanischer Einheiten ohne TCP auf Seite 418*
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*
Konfiguration von externen Achsen	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.41 listitem – Datenstruktur des Listenobjekts

Verwendung

`listitem` wird für das Definieren von Menüzeilen auf dem Benutzergerät wie beispielsweise dem FlexPendant verwendet, die Text mit optionalen kleinen Icons enthalten.

Beschreibung

Daten des Typs `listitem` ermöglichen dem Benutzer das Definieren von Menüzeilen für die Funktion `UIListView`.

Grundlegendes Beispiel

Das folgende Beispiel veranschaulicht den Datentyp `listitem` :

Beispiel 1:

```
CONST listitem list {3}:=[[stEmpty, "Item1"], [stEmpty, "Item2"],
     [stEmpty, "Item3"]];
```

Eine Menüliste mit Item1....Item3 zur Verwendung in Funktion `UIListView`.

Komponenten

Der Datentyp hat die folgenden Komponenten:

`image`

Datentyp: `string`

Der Pfad mit dem Dateinamen für das anzuzeigende Icon-Bild (in diesem Software-Release nicht implementiert).

Verwenden Sie die leere Zeichenfolge `""` oder `stEmpty`, wenn kein Icon angezeigt werden soll.

`text`

Datentyp: `string`

Der Text der anzuzeigenden Menüzeile.

Struktur

```
<dataobject of listitem>
  <image of string>
  <text of string>
```

Weitere Informationen

Information über	Siehe
Benutzerinteraktion "Listenansicht"	*UIListView - Anzeigen der Benutzerliste auf Seite 1596*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.42 loaddata - Lastdaten

Verwendung

`loaddata` wird zur Beschreibung von Lasten verwendet, die an der mechanischen Schnittstelle des Roboters (dem Befestigungsflansch) befestigt sind.

Lastdaten definieren gewöhnlich die Nutzlast oder Greiferlast (für Positionierer definiert durch die Instruktion `GripLoad` oder `MechUnitLoad`) des Roboters, d. h. die Last, die der Robotergreifer hält. `loaddata` werden auch als Teil der `tooldata` angegeben, um die Werkzeuglast zu beschreiben.

Beschreibung

Anhand bestimmter Lasten wird ein Modell der Roboterdynamik aufgestellt, damit die Roboterbewegungen ideal gesteuert werden.

WARNUNG

Die korrekte aktuelle Werkzeuglast sowie, falls vorhanden, die Nutzlast des Roboters müssen unbedingt definiert werden. Falsche Definitionen der Lastdaten können zu Überlastung der mechanischen Roboterstruktur führen. Es besteht zudem ein Risiko, dass die Geschwindigkeit im manuell reduzierten Geschwindigkeitsmodus überschritten werden kann.

Die Angabe falscher Lastdaten kann folgende Konsequenzen haben:

- Der Roboter verwendet möglicherweise nicht seine maximale Kapazität.
- Die Bahngenauigkeit wird beeinträchtigt und es besteht das Risiko des Überschwingens.
- Die mechanische Struktur kann überlastet werden.

Die Steuerung überwacht ununterbrochen die Last und schreibt ein Ereignisprotokoll, wenn die Last höher als erwartet ist. Dieses Ereignisprotokoll wird gespeichert und Steuerungsspeicher protokolliert.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `loaddata`:

Beispiel 1:

```
PERS loaddata piece1 := [ 5, [50, 0, 50], [1, 0, 0, 0], 0, 0, 0];
```

Die Nutzlast, die in der Abbildung *Vom Roboter gehaltenes Werkzeug auf Seite 1725* von einem vom Roboter gehaltenen Werkzeug bewegt wird, wird mit den folgenden Werten beschrieben:

- Gewicht `5` kg.
- Der Schwerpunkt liegt bei x = `50` , y =`0` und z = `50` mm im Werkzeug-Koordinatensystem.
- Die Nutzlast ist eine Punktlast.

Beispiel 2:

```
Set gripper;
WaitTime 0.3;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.42 loaddata - Lastdaten
RobotWare - OS
Fortsetzung

```
GripLoad piece1;
```

Aufnehmen der Nutzlast `piece1` wird gleichzeitig mit dem Greifen der Last definiert.

Beispiel 3

```
Reset gripper;
WaitTime 0.3;
GripLoad load0;
```

Ablegen der Nutzlast wird gleichzeitig mit dem Freigeben der Nutzlast definiert.

Beispiel 4:

```
PERS loaddata piece2 := [ 5, [50, 50, 50], [0, 0, 1, 0], 0, 0, 0];
PERS wobjdata wobj2 :=[ TRUE, TRUE, "", [ [0, 0, 0], [1, 0, 0 ,0]
     ], [ [50, -50, 200], [0.5, 0, -0.866 ,0] ] ];
```

Die Nutzlast, die in der Abbildung Stationäres Werkzeug von einem *Stationäres Werkzeug auf Seite 1726* bewegt wird, beschreiben folgende Werte für die `loaddata`:

- Gewicht 5 kg
- Der Schwerpunkt liegt bei x = `50`, y = `50` und z = `50` mm im Objekt-Koordinatensystem für Werkobjekt `wobj2`
- Das Nutzlast-Koordinatensystem/die Drehmomentachsen werden entsprechend dem Objekt-Koordinatensystem um 180° um Y" gedreht
- Die Nutzlast ist eine Punktlast.

Folgende Werte werden für die `wobjdata` verwendet:

- Der Roboter hält das Werkobjekt
- Das feste Anwender-Koordinatensystem wird verwendet, d. h. das Anwender-Koordinatensystem ist dasselbe wie das Handgelenk-Koordinatensystem.
- Das Objekt-Koordinatensystem wird -120° um Y gedreht und die Koordinaten seines Ursprungs sind im Anwender-Koordinatensystem x = `50`, y = `-50` und z = `200` mm.

Vordefinierte Daten

Die Last `load0` definiert eine Nutzlast, deren Masse gleich 0 kg, also gar keine Last ist. Diese Last wird als Argument in der Instruktion `GripLoad` und `MechUnitLoad` verwendet, um eine Nutzlast abzulegen.

Das Programm kann immer auf die Last `load0` zugreifen, aber die Last lässt sich nicht ändern (sie ist im Systemmodul `BASE` gespeichert).

```
PERS loaddata load0 := [ 0.001, [0, 0, 0.001], [1, 0, 0, 0], 0, 0
     , 0 ];
```

Komponenten

Hinweis

In dieser Beschreibung werden `loaddata` nur für die Verwendung als Nutzlast beschrieben. Für die Verwendung für die Werkzeuglast siehe *tooldata – Werkzeugdaten auf Seite 1823*.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`mass`

Datentyp: `num`

Die Masse (Gewicht) der Last in kg.

`cog`

center of gravity

Datentyp: `pos`

Der Schwerpunkt einer Nutzlast in mm im Werkzeug-Koordinatensystem, wenn der Roboter das Werkzeug hält. Wird ein stationäres Werkzeug verwendet, wird der Schwerpunkt für die vom Greifer gehaltene Nutzlast im Objekt-Koordinatensystem des Werkobjekt-Koordinatensystems, das vom Roboter bewegt wird.

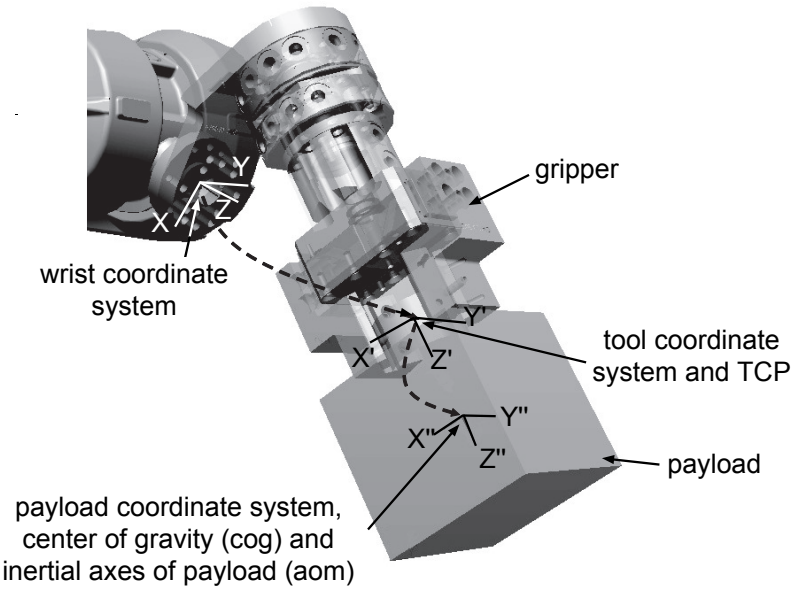
`aom`

axes of moment

Datentyp: `orient`

Die Orientierung der Drehmomentachsen. Dies sind die Hauptachsen für das Trägheitsmoment der Nutzlast mit dem Ursprung in `cog`. Hält der Roboter das Werkzeug, werden die Drehmomentachsen im Werkzeug-Koordinatensystem ausgedrückt.

Die Abbildung zeigt den Schwerpunkt und die Trägheitsachsen der Nutzlast.



xx1100000515

Figure 3.1: Vom Roboter gehaltenes Werkzeug

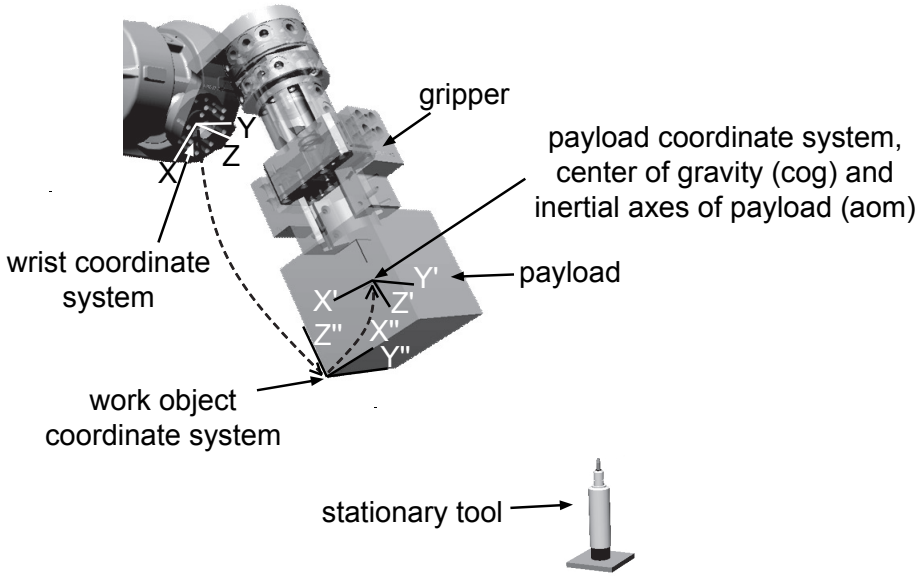
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Hinweis

Wenn *PayloadsInWristCoords* verwendet wird, werden die Drehmomentachsen für die Nutzlast für das Werkzeug im Handgelenk-Koordinatensystem ausgedrückt. Weitere Informationen siehe *Technisches Referenzhandbuch - Systemparameter*, Abschnitt *PayloadsInWristCoords*.

Die Drehmomentachsen werden im Objekt-Koordinatensystem ausgedrückt, wenn ein stationäres Werkzeug verwendet wird.



xx1100000516

Figure 3.2: Stationäres Werkzeug

Hinweis

Wenn *PayloadsInWristCoords* oder *StationaryPayLoadMode* verwendet wird, werden die Drehmomentachsen für die Nutzlast für das stationäre Werkzeug im Handgelenk-Koordinatensystem ausgedrückt. Weitere Informationen siehe *Technisches Referenzhandbuch - Systemparameter*, Abschnitt *PayloadsInWristCoords* und *StationaryPayLoadMode*.

`ix`

inertia x

Datentyp: `num`

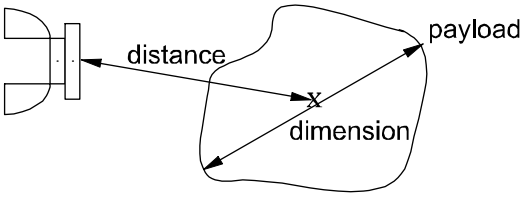
Das Trägheitsmoment der Last um die x-Achse, ausgedrückt in kgm^2.

Die korrekte Definition des Trägheitsmoments ermöglicht die optimale Verwendung des Bahnplaners und der Achsensteuerung. Dies ist bei der Handhabung großer Blechteile usw. von besonderer Wichtigkeit. Alle Massenträgheitsmomente `ix`, `iy` und `iz`, die gleich 0 kgm^2 sind, weisen auf eine Punktmasse hin.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Gewöhnlich müssen Massenträgheitsmomente nur dann definiert werden, wenn der Abstand zwischen dem Befestigungsflansch und dem Schwerpunkt kleiner als die Abmessungen der Last ist (siehe folgende Abbildung).



xx0500002372

`iy`

inertia y

Datentyp: `num`

Das Trägheitsmoment der Last um die y-Achse, ausgedrückt in kgm^2.

Weitere Informationen finden Sie im `ix`.

`iz`

inertia z

Datentyp: `num`

Das Trägheitsmoment der Last um die z-Achse, ausgedrückt in kgm^2.

Weitere Informationen finden Sie im `ix`.

Einschränkungen

Die Nutzlast darf nur als persistente Variable (`PERS`) und nicht innerhalb einer Routine definiert werden. Aktuelle Werte werden dann gespeichert, wenn Sie das Programm speichern, und werden beim Laden wieder abgerufen.

Argumente des Typs `loaddata` in der Instruktion `GripLoad` und `MechUnitLoad` dürfen nur eine vollständige Persistente sein (kein Datenfeldelement und keine Datensatzkomponente).

Struktur

```
<dataobject of loaddata>
  <mass of num>
  <cog of pos>
    <x of num>
    <y of num>
    <z of num>
  <aom of orient>
    <q1 of num>
    <q2 of num>
    <q3 of num>
    <q4 of num>
  <ix of num>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.42 loaddata - Lastdaten

RobotWare - OS

Fortsetzung

```
                    <iy of num>
                    <iz of num>
```

Weitere Informationen

Information über	Siehe
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*
Definition von Werkzeuglasten	*tooldata – Werkzeugdaten auf Seite 1823*
Definieren der Nutzlast für Roboter	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
Definieren der Nutzlast für mechanische Einheiten	*MechUnitLoad - Definieren einer Nutzlast für eine mechanische Einheit auf Seite 372*
Lastidentifikation der Werkzeuglast, Nutzlast oder Armlast	*Bedienungsanleitung - IRC5 mit FlexPendant*
Definition von Armlasten	*Technisches Referenzhandbuch - Systemparameter*, siehe *Definieren von Armlasten*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*
`PayloadsInWristCoords` `StationaryPayLoadMode`	*Technisches Referenzhandbuch - Systemparameter*, Parametergruppe *Controller*, Typ *General Rapid*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.43 loadidnum – Lastidentifikationstyp

Verwendung

`loadidnum` stellt eine Ganzzahl mit einer symbolischen Konstanten dar.

Beschreibung

Eine `loadidnum`-Konstante ist als Argument der Instruktion `LoadId` bei der Lastidentifikation eines Werkzeugs oder einer Nutzlast vorgesehen. Betrachten Sie das folgende Beispiel.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `loadidnum`:

Beispiel 1:

```
! Load modules into the system
Load \Dynamic, "RELEASE:/system/mockit.sys";
Load \Dynamic, "RELEASE:/system/mockit1.sys";
%"LoadId"% TOOL_LOAD_ID, MASS_WITH_AX3, gun1;
```

Lastidentifikation des Werkzeugs `gun1` mit Identifikation der Masse bei Bewegung der Roboterachse 3 unter Verwendung der vordefinierten Konstante `MASS_WITH_AX3` des Datentyps `loadidnum`.

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `loadidnum` sind vordefiniert und werden in der Instruktion `LoadId` als Argumente verwendet.

Wert	Symbolische Konstante	Kommentar
1	`MASS_KNOWN`	Bekannte Masse des Werkzeugs bzw. der Nutzlast
2	`MASS_WITH_AX3`	Unbekannte Masse der Werkzeug- oder Nutzlast. Masse wird bei Bewegung der Achse 3 festgestellt.

Eigenschaften

`loadidnum` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Vordefiniertes Programm LoadIdentify	*Bedienungsanleitung - IRC5 mit FlexPendant*, Abschnitt *Programming and testing - Service routines- LoadIdentify, load identification and service routines*
Gültiger Robotertyp	*ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation auf Seite 1424*
Gültige Roboterposition	*ParIdPosValid - Gültige Roboterposition für Parameteridentifikation auf Seite 1421*
Lastidentifikation mit komplettem Beispiel	*LoadId - Lastidentifikation der Werkzeug- oder Nutzlast auf Seite 351*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.44 loadsession – Ladevorgang für ein Programm

Verwendung

`loadsession` wird verwendet, um verschiedene Ladevorgänge von RAPID-Programmmodulen zu definieren.

Beschreibung

Daten des Typs `loadsession` werden in den Instruktionen `StartLoad` und `WaitLoad` verwendet, um den Ladevorgang zu identifizieren. `loadsession` enthält nur einen Verweis auf den Ladevorgang.

Eigenschaften

`loadsession` ist ein Datentyp ohne einen Wert (Non-Value) und kann nicht in wertorientierten Operationen verwendet werden.

Weitere Informationen

Information über	Siehe
Laden von Programmmodulen während der Abarbeitung	*StartLoad – Laden eines Programmmoduls während der Abarbeitung auf Seite 802* *WaitLoad – Das geladene Modul mit der Task verbinden auf Seite 1078*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.45 mecunit – Mechanische Einheit

Verwendung

`mecunit` dient der Definition der verschiedenen mechanischen Einheiten, die vom Programm gesteuert werden können bzw. auf die das Programm zugreifen kann.

Die Namen der mechanischen Einheiten werden in den Systemparametern definiert und müssen daher nicht im Programm festgelegt werden.

Beschreibung

Daten des Typs `mecunit` enthalten nur einen Verweis auf die mechanische Einheit.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `mecunit`:

Beispiel 1:

```
IF TaskRunRob() THEN
  IndReset ROB_ID, 6;
ENDIF
```

Wenn die aktuelle Programmtask einen Roboter steuert, setzen Sie Achse `6` für einen Pick-and-Place-Roboter zurück.

Vordefinierte Daten

Alle mechanischen Einheiten, die in den Systemparametern definiert sind, sind in jeder Programmtask vordefiniert. Es können jedoch nur die mechanischen Einheiten für die Durchführung von Steuerungsoperationen verwendet werden, die vom aktuellen Programmtask gesteuert werden (definiert in den Systemparametern *Controller/Task/Use Mechanical Unit Group*).

Davon abgesehen ist die vordefinierte Variable `ROB_ID` vom Datentyp `mecunit` in jeder Programmtask verfügbar. Wenn eine aktuelle Programmtask einen Roboter steuert, enthält die Alias-Variable `ROB_ID` einen Verweis auf Roboter `ROB_1` bis `ROB_6`, der zur Steuerung der Operationen am Roboter verwendet werden kann. Die Variable `ROB_ID` ist ungültig, wenn die aktuelle Programmtask keinen Roboter steuert.

Einschränkungen

Daten des Typs `mecunit` dürfen nicht im Programm definiert werden. Anderenfalls wird eine Fehlermeldung angezeigt, sobald eine Instruktion oder Funktion abgearbeitet wird, die auf dieses `mecunit` verweist. Der Datentyp kann hingegen beim Deklarieren einer Routine als Parameter benutzt werden.

Eigenschaften

`mecunit` ist ein Datentyp *ohne einen Wert* (Non-Value). Das heißt, Daten dieses Typs erlauben keine wertorientierten Operationen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.45 mecunit – Mechanische Einheit
RobotWare - OS
Fortsetzung

Weitere Informationen

Information über	Siehe
Prüfen, ob die Task einen Roboter steuert	*TaskRunRob - Überprüfen, ob die Task einen Roboter steuert auf Seite 1542*
Prüfen, ob die Task eine mechanische Einheit ausführt	*TaskRunMec - Überprüfen, ob die Task eine mechanische Einheit steuert auf Seite 1541*
Abrufen des Namens von mechanischen Einheiten im System	*GetNextMechUnit – Abrufen des Namens und der Daten für mechanische Einheiten auf Seite 1325*
Aktivieren/Deaktivieren mechanischer Einheiten	*ActUnit - Aktivieren einer mechanischen Einheit auf Seite 30* *DeactUnit - Deaktivieren einer mechanischen Einheit auf Seite 187*
Konfiguration von mechanischen Einheiten	*Technisches Referenzhandbuch - Systemparameter*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.46 motsetdata - Daten für Bewegungseinstellungen

Verwendung

`motsetdata`wird zur Definition einer Reihe von Bewegungseinstellungen verwendet, die alle Bewegungsinstruktionen im Programm beeinflussen:

- Höchstgeschwindigkeit und Geschwindigkeitsoverride
- Beschleunigungsdaten
- Verhalten in der Nähe von singulären Punkten
- Verwaltung verschiedener Roboterkonfigurationen
- Override der Bahnauflösung
- Bewegungsüberwachung
- Beschränkung der Beschleunigung/Verzögerung
- Werkzeugumorientierung auf kreisförmiger Bahn
- Aktivierung und Deaktivierung des Ereignispuffers

Dieser Datentyp muss normalerweise nicht verwendet werden, da diese Einstellungen nur über die Instruktionen `VelSet`, `AccSet`, `SingArea`, `ConfJ`, `ConfL`, `PathResol`, `MotionSup`, `PathAccLim`, `CirPathMode`, `WorldAccLim`, `ActEventBuffer`, `DeactEventBuffer` und `CornerPathWarning`. eingestellt werden.

Auf die aktuellen Werte dieser Bewegungseinstellungen können Sie mit der Systemvariable `C_MOTSET` zugreifen.

Beschreibung

Die aktuellen Bewegungseinstellungen (in der Systemvariable `C_MOTSET`) beeinflussen alle Bewegungen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `motsetdata`:

Beispiel 1:

```
IF C_MOTSET.vel.oride > 50 THEN
  ...
ELSE
  ...
ENDIF
```

Verschiedene Teile des Programms werden abhängig vom aktuellen Geschwindigkeitsoverride abgearbeitet.

Vordefinierte Daten

`C_MOTSET` beschreibt die aktuellen Bewegungseinstellungen des Roboters und das Programm kann immer darauf zugreifen. `C_MOTSET` hingegen lässt sich nur über eine Reihe von Instruktionen ändern, nicht durch Zuweisung.

Die folgenden Standardwerte für Bewegungsparameter werden gesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.46 motsetdata - Daten für Bewegungseinstellungen
RobotWare - OS
Fortsetzung

- bei Beginn der Programmabarbeitung vom Anfang
- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

```
VAR motsetdata C_MOTSET := [
  [ 100, 5000 ],-> veldata
  [ 100, 100, 100 ],-> accdata
  [ FALSE, FALSE, FALSE, TRUE ],-> singdata
  [ TRUE, TRUE, 30, 45, 90 ]-> confsupdata
  100,-> path resolution
  TRUE,-> motionsup
  100,-> tunevalue
  TRUE,-> backoffaftercoll
  FALSE,-> acclim
  -1,-> accmax
  FALSE,-> decellim
  -1,-> decelmax
  0,-> cirpathreori
  FALSE,-> worldacclim
  -1,-> worldaccmax
  TRUE,-> evtbufferact
  FALSE];-> corner_path_warn_suppress
```

Hinweis

Die maximale TCP-Geschwindigkeit für den verwendeten Robotertyp kann in der Konfiguration des Systemparameters *Motion*, Typ *Motion Planner* und Attribut *Linear Max Speed*, geändert werden. Die RAPID-Funktion `MaxRobSpeed` gibt denselben Wert zurück.

Komponenten

Hinweis

Einige Komponenten werden in der Struktur vorbereitet, werden aber derzeit nicht in die entsprechenden Instruktionen implementiert.

`vel.oride`

Datentyp: `veldata/num`

Geschwindigkeitsoverride als Prozentsatz des programmierten Werts.

`vel.max`

Datentyp: `veldata/num`

Höchstgeschwindigkeit in mm/s.

`acc.acc`

Datentyp: `accdata/num`

Beschleunigung und Verzögerung als Prozentsatz der normalen Werte.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

`acc.ramp`

Datentyp: `accdata/num`

Anstiegswert von Beschleunigung und Verzögerung als Prozentsatz der normalen Werte.

`acc.finepramp`

Datentyp: `accdata/num`

Wert, zu welchem die Verzögerung als Prozentsatz der normalen Werte abnimmt, wenn sich der Roboter zu einem Feinpunkt hin verzögert.

`sing.wrist`

Datentyp: `singdata/bool`

Die Orientierung des Werkzeugs darf geringfügig abweichen, um Singularität des Handgelenks zu verhindern.

`sing.lockaxis4`

Datentyp: `singdata/bool`

Sperren der Achse 4 bei einem sechsachsigen Roboter bis 0 oder ±180 Grad, um Singularitätsprobleme zu vermeiden, wenn sich Achse 5 dem Wert 0 nähert.

`sing.arm`

Datentyp: `singdata/bool`

Die Orientierung des Werkzeugs darf geringfügig abweichen, um Singularität des Arms zu verhindern (nicht implementiert).

`sing.base`

Datentyp: `singdata/bool`

Die Orientierung des Werkzeugs darf nicht abweichen.

`conf.jsup`

Datentyp: `confsupdata/bool`

Während der Achsenbewegung erreicht der Roboter die programmierte Roboterkonfiguration.

`conf.lsup`

Datentyp: `confsupdata/bool`

Überwachung der Achsenkonfiguration ist bei der linearen und kreisförmigen Bewegung aktiv.

`conf.ax1`

Datentyp: `confsupdata/num`

Maximal zulässige Abweichung in Grad für Achse 1 (nicht implementiert).

`conf.ax4`

Datentyp: `confsupdata/num`

Maximal zulässige Abweichung in Grad für Achse 4 (nicht implementiert).

`conf.ax6`

Datentyp: `confsupdata/num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.46 motsetdata - Daten für Bewegungseinstellungen
RobotWare - OS
Fortsetzung

Maximal zulässige Abweichung in Grad für Achse 6 (nicht implementiert).

`pathresol`

Datentyp: `num`

Aktueller Override in Prozentsatz der konfigurierten Bahnauflösung.

`motionsup`

Datentyp: `bool`

Spiegelt den RAPID-Status (`TRUE` = Ein und `FALSE` = Aus) der Funktion zur Bewegungsüberwachung.

`tunevalue`

Datentyp: `num`

Aktueller RAPID-Override als Prozentsatz des konfigurierten Abstimmungswerts (tunevalue) für die Bewegungsüberwachung.

`backoffaftercoll`

Datentyp: `bool`

RAPID-Status der Rückwärtsbewegung spiegeln, um bei einer Bewegungskollision vorhandene Restenergien abzubauen:

`TRUE` = Rückwärtsbewegung durchführen, um bei einer Bewegungskollision vorhandene Restenergien abzubauen

`FALSE` = keine Rückwärtsbewegung bei einer Bewegungskollision

`acclim`

Datentyp: `bool`

Beschränkung der Werkzeugbeschleunigung entlang der Bahn. (`TRUE` = Beschränkung und `FALSE` = keine Beschränkung).

`accmax`

Datentyp: `num`

Beschränkung der TCP-Beschleunigung in m/s ^{2}. Wenn acclim gleich `FALSE` ist, wird der Wert stets auf -1 gesetzt.

`decellim`

Datentyp: `bool`

Beschränkung der Werkzeugverzögerung entlang der Bahn. (`TRUE` = Beschränkung und `FALSE` = keine Beschränkung).

`decelmax`

Datentyp: `num`

Beschränkung der TCP-Verzögerung in m/s ^{2}. Wenn `decellim` gleich `FALSE` ist, wird der Wert stets auf -1 gesetzt.

`cirpathreori`

Datentyp: `num`

Werkzeugumorientierung auf kreisförmiger Bahn:

0 = Standardmethode mit Interpolation in Bahn-Koordinatensystem

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

1 = Modifizierte Methode mit Interpolation in Objekt-Koordinatensystem

2 = Modifizierte Methode mit programmierter Werkzeugorientierung in `CirPoint`

`worldacclim`

Datentyp: `bool`

Beschränkung der Beschleunigung in Welt-Koordinatensystem. (`TRUE` = Beschränkung und `FALSE` = keine Beschränkung).

`worldaccmax`

Datentyp: `num`

Beschränkung der Beschleunigung in Welt-Koordinatensystem in m/s ^{2}. Wenn `worldacclim` FALSE ist, wird der Wert stets auf -1 gesetzt.

`evtbufferact`

Datentyp: `bool`

Ereignispuffer aktiv oder inaktiv. (`TRUE` = Ereignispuffer aktiv und `FALSE` = Ereignispuffer inaktiv).

`corner_path_warn_suppress`

Datentyp: `bool`

Zonenbahnwarnung kann gemeldet werden. `TRUE` = Zonenbahnwarnung wird unterdrückt, `FALSE` = Zonenbahnwarnung wird nicht unterdrückt.

Einschränkungen

Nur eine einzige der Komponenten `sing.wrist`, `sing.arm` oder `sing.base` darf den Wert TRUE haben.

Struktur

```
<dataobject of motsetdata>
```

```
<vel of veldata>
  <oride of num>
  <max of num>
```

Beeinflusst von der Instruktion `VelSet`

```
<acc of accdata>
  <acc of num>
  <ramp of num>
  <finepramp of num>
```

Beeinflusst von der Instruktion `AccSet`

```
<sing of singdata>
  <wrist of bool>
  <lockaxis4 of bool>
  <arm of bool>
  <base of bool>
```

Beeinflusst von der Instruktion `SingArea`

```
<conf of confsupdata>
  <jsup of bool>
  <lsup of bool>
  <ax1 of num>
  <ax4 of num>
  <ax6 of num>
```

Beeinflusst von den Instruktionen `ConfJ` und `ConfL`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.46 motsetdata - Daten für Bewegungseinstellungen

RobotWare - OS

Fortsetzung

`<pathresol of num>`	Beeinflusst von der Instruktion `PathResol`
`<motionsup of bool>`	Beeinflusst von der Instruktion `MotionSup`
`<tunevalue of num>`	Beeinflusst von der Instruktion `MotionSup`
`<backoffaftercoll of bool>`	Beeinflusst von der Instruktion `MotionSup`
`<acclim of bool>`	Beeinflusst von der Instruktion `PathAccLim`
`<accmax of num>`	Beeinflusst von der Instruktion `PathAccLim`
`<decellim of bool>`	Beeinflusst von der Instruktion `PathAccLim`
`<decelmax of num>`	Beeinflusst von der Instruktion `PathAccLim`
`<cirpathreori of num>`	Beeinflusst von der Instruktion `CirPathMode`
`<worldacclim of bool>`	Beeinflusst von der Instruktion `WorldAccLim`
`<worldaccmax of num>`	Beeinflusst von der Instruktion `WorldAccLim`
`<evtbufferact of bool>`	Beeinflusst von den Instruktionen `ActEventBuffer` und `DeactEventBuffer`
`<corner_path_warn_suppress of bool>`	Beeinflusst von der Instruktion `CornerPathWarning`

Weitere Informationen

Information über	Siehe
Reduktion der Beschleunigung	*AccSet - Verringern der Beschleunigung auf Seite 25*
Aktivierung des Ereignispuffers	*ActEventBuffer – Aktivierung des Ereignispuffers auf Seite 28*
Werkzeugumorientierung auf kreisförmiger Bahn	*CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn auf Seite 133*
Roboterkonfiguration bei Achsenbewegung	*ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung auf Seite 156*
Roboterkonfiguration bei linearer Bewegung	*ConfL - Überwachen der Konfiguration während linearer Bewegung auf Seite 158*
Zonenbahnwarnungen unterdrücken	*CornerPathWarning – Zonenbahnwarnungen anzeigen oder verbergen auf Seite 174*
Deaktivierung des Ereignispuffers	*DeactEventBuffer – Deaktivierung des Ereignispuffers auf Seite 185*
Deaktivieren oder Aktivieren der Bewegungsüberwachungsfunktion	*MotionSup - Deaktivieren/Aktivieren der Bewegungsüberwachung auf Seite 379*
Beschränkung der Beschleunigung entlang der Bahn	*PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn auf Seite 507*
Überschreiben der konfigurierten geometrischen Bahn-Abtastrate	*PathResol - Override der Bahnauflösung auf Seite 534*
Definieren der Interpolation in der Nähe singulärer Punkte	*SingArea - Definieren der Interpolation in der Nähe singulärer Punkte auf Seite 736*
Definieren der Höchstgeschwindigkeit	*VelSet - Ändern der programmierten Geschwindigkeit auf Seite 1038*
Steuern der Beschleunigung im Welt-Koordinatensystem	*WorldAccLim - Steuern der Beschleunigung im Welt-Koordinatensystem auf Seite 1112*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Instruktionen zur Einstellung von Bewegungsparametern	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion settings*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.47 num - Numerische Werte

Verwendung

`num` wird für numerische Werte wie z. B. Zähler verwendet.

Beschreibung

Der Datentyp `num` kann folgende Werte haben:

- eine Ganzzahl, z. B. -5,
- eine Dezimalzahl, z. B. 3,45.

Er lässt sich auch in Exponentialschreibweise ausdrücken, z. B. 2E3 (= 2*10^3 = 2000), 2,5E-2 (= 0,025).

Ganzzahlen zwischen -8388607 und +8388608 werden immer als exakte Ganzzahlen gespeichert.

Dezimalzahlen sind nur Näherungswerte und sollten deshalb nicht in Vergleichen mit *ist gleich* oder *ist ungleich* verwendet werden. Für Divisionen und Operationen mit Dezimalzahlen ist auch das Ergebnis eine Dezimalzahl, d. h. keine exakte Ganzzahl. Beispiel:

```
a := 10;
b := 5;
IF a/b=2 THEN

...
```

Da das Ergebnis von a/b keine Ganzzahl ist, ist diese Bedingung nicht notwendigerweise erfüllt.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `num`:

Beispiel 1:

```
VAR num reg1;
...
reg1 := 3;
```

`reg1` wird der Wert `3` zugewiesen.

Beispiel 2:

```
a := 10 DIV 3;
b := 10 MOD 3;
```

Ganzzahldivision, wobei `a` eine Ganzzahl (=3) und `b` der Rest (=1) zugewiesen wird.

Vordefinierte Daten

Einige Daten sind im System vordefiniert. Beispielsweise ist die Konstante Pi (π) definiert.

CONST num pi := 3.1415926;

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

Literale zwischen -8388607 und 8388608 in Verbindung mit einer `num`-Variablen werden als exakte Ganzzahlen gespeichert.

Wenn ein als `dnum` interpretierter literaler Wert als `num` zugeordnet/verwendet wird, wird er automatisch zu einem `num`.

Weitere Informationen

Information über	Siehe
Numerische Werte mit dem Datentyp dnum	*dnum – Doppelte numerische Werte auf Seite 1690*
Numerische Ausdrücke	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic RAPID programming - Expressions*
Operationen mit numerischen Werten	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic RAPID programming - Expressions*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.48 opcalc - Arithmetischer Operator

Verwendung

Mit `opcalc` wird ein arithmetischer Operator in Argumenten für RAPID-Funktionen oder -Instruktionen dargestellt.

Beschreibung

Eine `opcalc`-Konstante dient zum Definieren des Typs der arithmetischen Operation.

Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `opcalc`:

Beispiel 1:

```
res := StrDigCalc(str1, OpAdd, str2);
```

`res` wird dem Ergebnis der Additionsoperation für die Werte zugewiesen, die von den Zeichenfolgen `str1` und `str2` dargestellt werden. `OpAdd`ist vom Datentyp `opcalc`.

Vordefinierte Daten

Die folgenden symbolischen Konstanten vom Datentyp `opcalc` sind vordefiniert und können für die Definition des Typs der verwendeten arithmetischen Operation, z. B. in der Funktion `StrDigCalc`, verwendet werden.

Konstante	Wert	Kommentar
OpAdd	1	Addition (+)
OpSub	2	Subtraktion (-)
OpMult	3	Multiplikation (*)
OpDiv	4	Division (/)
OpMod	5	Modulo (%I)

Eigenschaften

`opcalc` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*
Arithmetische Operationen an Zeichenfolgen	*StrDigCalc - Arithmetische Operationen mit dem Datentyp stringdig auf Seite 1515*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.49 opnum - Vergleichsoperator

Verwendung

`opnum` wird für einen Operator verwendet, der Vergleichen in Argumenten von RAPID-Funktionen oder -Instruktionen dient.

Beschreibung

Eine `opnum`-Konstante ist für die Definition des Vergleichstyps vorgesehen, wenn Werte in generischen Instruktionen geprüft werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `opnum`:

Beispiel 1:

```
TriggCheckIO checkgrip, 100, airok, EQ, 1, intno1;
```

Vordefinierte Daten

Die folgenden symbolischen Konstanten vom Datentyp `opnum` sind vordefiniert und können für die Definition des Typs des verwendeten Vergleichs, z. B. in der Funktion `TriggCheckIO`, verwendet werden.

Wert	Symbolische Konstante	Kommentar
1	`LT`	Kleiner als
2	`LTEQ`	Kleiner oder gleich
3	`EQ`	Gleich
4	`NOTEQ`	Ungleich
5	`GTEQ`	Größer oder gleich
6	`GT`	Greater Than

Eigenschaften

`opnum` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*
Definieren einer positionsabhängigen E/A-Prüfung	*TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.50 orient – Ausrichtung

Verwendung

`orient` wird für Orientierungen (z. B. die Orientierung eines Werkzeugs) und Drehungen (z. B. die Drehung eines Koordinatensystems) benutzt.

Beschreibung

Die Orientierung wird als Quaternion beschrieben, das aus vier Komponenten besteht: `q1`, `q2`, `q3` und `q4`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `orient`:

Beispiel 1:

```
VAR orient orient1;
.
orient1 := [1, 0, 0, 0];
```

Der Orientierung `orient1` wird der Wert q1=1, q2-q4=0 zugewiesen; d. h., es erfolgt keine Drehung.

Komponenten

Der Datentyp `orient` hat die folgenden Komponenten:

`q1`

Datentyp: `num`

Quaternion 1.

`q2`

Datentyp: `num`

Quaternion 2.

`q3`

Datentyp: `num`

Quaternion 3.

`q4`

Datentyp: `num`

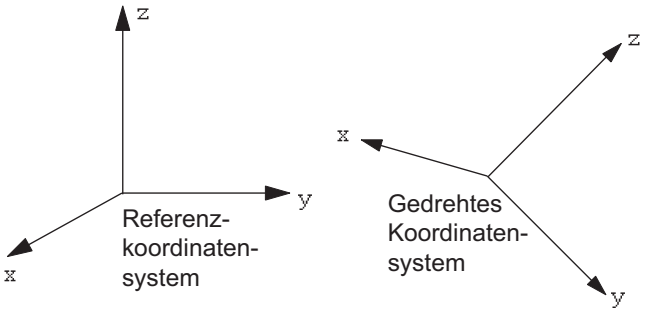
Quaternion 4.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Was ist ein Quaternion?

Die Orientierung eines Koordinatensystems (z. B. für ein Werkzeug) wird als Rotationsmatrix beschrieben, die die Richtung der Achsen des Koordinatensystems in Relation zu einem Referenz-Koordinatensystem beschreibt (siehe folgende Abbildung).



xx0500002376

Die Achsen des gedrehten Koordinatensystems (x, y, z) sind Vektoren, die sich wie folgt in einem Referenz-Koordinatensystem ausdrücken lassen:

x = (x1, x2, x3)

y = (y1, y2, y3)

z = (z1, z2, z3)

Das bedeutet, dass die x-Komponente des x-Vektors im Referenz-Koordinatensystem gleich x1, die y-Komponente gleich x2 usw. sein wird.

Diese drei Vektoren können zusammen in eine Matrix, eine Rotationsmatrix, gesetzt werden, in der jeder der Vektoren eine Spalte bildet:

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}$$

xx0500002381

Ein Quaternion ist einfach eine kompaktere Methode zur Beschreibung dieser Rotationsmatrix. Die Quaternionen werden auf der Basis der Elemente der Rotationsmatrix berechnet:

q1 = $\frac{\sqrt{x_1+y_2+z_3+1}}{2}$	
q2 = $\frac{\sqrt{x_1-y_2-z_3+1}}{2}$	sign q2 = sign (y 3-z 2)
q3 = $\frac{\sqrt{y_2-x_1-z_3+1}}{2}$	sign q3 = sign (z 1-x 3)
q4 = $\frac{\sqrt{z_3-x_1-y_2+1}}{2}$	sign q4 = sign (x 2-y 1)

Fortsetzung auf nächster Seite

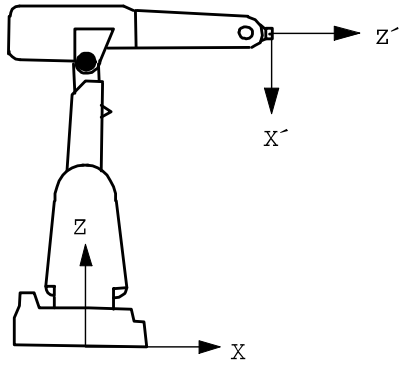
© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.50 orient – Ausrichtung
RobotWare - OS
Fortsetzung

Beispiel 1:

Ein Werkzeug ist so orientiert, dass seine Z'-Achse direkt nach vorne weist (in dieselbe Richtung wie die X-Achse des Basis-Koordinatensystems). Die Y'-Achse des Werkzeugs entspricht der Y-Achse des Basis-Koordinatensystems (siehe folgende Abbildung). Wie ist die Orientierung des Werkzeugs in den Positionsdaten definiert (`robtarget`)?

Die Orientierung des Werkzeugs an einer programmierten Position wird gewöhnlich auf das Koordinatensystem des Werkobjekts bezogen. In diesem Beispiel wird kein Werkobjekt verwendet und das Basis-Koordinatensystem ist gleich dem Welt-Koordinatensystem. Daher bezieht sich die Orientierung auf das Basis-Koordinatensystem.



xx0500002377

Die Achsen werden dann wie folgt angegeben:

x' = -z = (0, 0, -1)

y' = y = (0, 1, 0)

z' = x = (1, 0, 0)

Dies entspricht der folgenden Rotationsmatrix:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

xx0500002388

Die Rotationsmatrix liefert einen entsprechenden Quaternion:

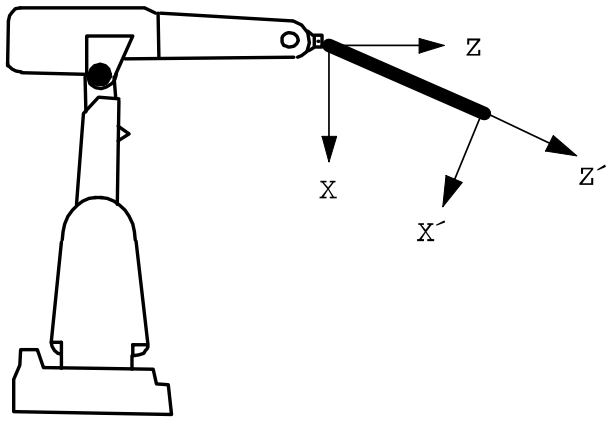
q1 = $\frac{\sqrt{0+1+0+1}}{2}$ = $\frac{\sqrt{2}}{2}$ = 0,707	
q2 = $\frac{\sqrt{0-1-0+1}}{2}$ = 0	
q3 = $\frac{\sqrt{1-0-0+1}}{2}$ = $\frac{\sqrt{2}}{2}$ = 0,707	sign q3 = sign (1+1) = +
q4 = $\frac{\sqrt{0-0-1+1}}{2}$ = 0	

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beispiel 2:

Die Richtung des Werkzeugs wird 30° um die X'- und Z'-Achsen in Bezug auf das Handgelenk-Koordinatensystem gedreht (siehe folgende Abbildung). Wie ist die Orientierung des Werkzeugs in den Werkzeugdaten definiert?



xx0500002378

Die Achsen werden dann wie folgt angegeben:

x' = (cos30°, 0, -sin30°)

y' = (0, 1, 0)

z' = (sin30°, 0, cos30°)

Dies entspricht der folgenden Rotationsmatrix:

$$\begin{bmatrix} \cos 30° & 0 & \sin 30° \\ 0 & 1 & 0 \\ -\sin 30° & 0 & \cos 30° \end{bmatrix}$$

xx0500002393

Die Rotationsmatrix liefert einen entsprechenden Quaternion:

q1 = $\frac{\sqrt{cos30°+1+cos30°+1}}{2}$ = 0,965926	
q2 = $\frac{\sqrt{cos30°-1-cos30°+1}}{2}$ = 0	
q3 = $\frac{\sqrt{1-cos30°-cos30°+1}}{2}$ = 0,258819	sign q3 = sign (sin30°+sin30°) = +
q4 = $\frac{\sqrt{cos30°-cos30°-1+1}}{2}$ = 0	

Struktur

```
< dataobject of orient >
  < q1 of num >
  < q2 of num >
  < q3 of num >
  < q4 of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.50 orient – Ausrichtung
RobotWare - OS
Fortsetzung

Einschränkungen

Die Orientierungen muss normalisiert werden, d. h., die Summe der Quadrate muss gleich 1 sein:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

Weitere Informationen

Information über	Siehe
Operationen mit Orientierungen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Expressions*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.51 paridnum - Parameteridentifikationstyp

Verwendung

`paridnum` stellt eine Ganzzahl mit einer symbolischen Konstanten dar.

Beschreibung

Eine `paridnum`-Konstante ist für die Parameteridentifikation wie z. B. Lastidentifikation von Werkzeug- oder Nutzlast oder externer Manipulatorlast vorgesehen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `paridnum`:

Beispiel 1:

```
TEST ParIdRobValid (TOOL_LOAD_ID)
  CASE ROB_LOAD_VAL:
  ! Possible to do load identification of tool in actual robot type
  ...
  CASE ROB_LM1_LOAD_VAL:
  ! Only possible to do load identification of tool with
  ! IRB 6400FHD if actual load < 200 kg
  ...
  CASE ROB_NOT_LOAD_VAL:
  ! Not possible to do load identification of tool in actual robot
       type
  ...
ENDTEST
```

Verwendung der vordefinierten Konstante `TOOL_LOAD_ID` des Datentyps `paridnum`.

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `paridnum` sind vordefiniert und werden in der Instruktion `ParIdRobValid`, `ParIdPosValid`, `LoadId` und `ManLoadIdProc` als Argumente verwendet.

Wert	Symbolische Konstante	Kommentar
1	`TOOL_LOAD_ID`	Identifizieren der Werkzeuglast
2	`PAY_LOAD_ID`	Identifizieren der Nutzlast (siehe Instruktion `GripLoad`)
3	`IRBP_K`	Identifizieren der Last des externen Manipulators IRBP K
4	`IRBP_L`	Identifizieren der Last des externen Manipulators IRBP L
4	`IRBP_C`	Identifizieren der Last des externen Manipulators IRBP C
4	`IRBP_C_INDEX`	Identifizieren der Last des externen Manipulators IRBP C_INDEX
4	`IRBP_T`	Identifizieren der Last des externen Manipulators IRBP T

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.51 paridnum - Parameteridentifikationstyp

RobotWare - OS

Fortsetzung

Wert	Symbolische Konstante	Kommentar
5	IRBP_R	Identifizieren der Last des externen Manipulators IRBP R
6	IRBP_A	Identifizieren der Last des externen Manipulators IRBP A
6	IRBP_B	Identifizieren der Last des externen Manipulators IRBP B
6	IRBP_D	Identifizieren der Last des externen Manipulators IRBP D

Hinweis

Nur TOOL_LOAD_ID und PAY_LOAD_ID werden in benutzerdefinierten RAPID-Programmen zur Identifizierung der Werkzeuglast bzw. der Nutzlast für den Roboter verwendet.

Eigenschaften

paridnum ist ein Alias-Datentyp für num und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Vordefiniertes Programm Load Identify	*Bedienungsanleitung - IRC5 mit FlexPendant*
Gültiger Robotertyp	*ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation auf Seite 1424*
Gültige Roboterposition	*ParIdPosValid - Gültige Roboterposition für Parameteridentifikation auf Seite 1421*
Lastidentifikation mit komplettem Beispiel	*LoadId - Lastidentifikation der Werkzeug- oder Nutzlast auf Seite 351*
Lastidentifikation von externen Manipulatoren	*ManLoadIdProc - Lastidentifikation des Manipulators IRBP auf Seite 359*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.52 paridvalidnum - Ergebnis von ParIdRobValid

Verwendung

`paridvalidnum` stellt eine Ganzzahl mit einer symbolischen Konstanten dar.

Beschreibung

Eine `paridvalidnum`-Konstante dient bei der Parameteridentifikation wie z. B. der Lastidentifikation eines Werkzeugs oder einer Nutzlast der Prüfung der zurückgegebenen Werte aus der Funktion `ParIdRobValid`.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `paridvalidnum`:

```
TEST ParIdRobValid (PAY_LOAD_ID)
  CASE ROB_LOAD_VAL:
  ! Possible to do load identification of payload in actual robot
  ! type
  ...
  CASE ROB_LM1_LOAD_VAL:
  ! Only possible to do load identification of payload
  ! with IRB 6400FHD if actual load < 200 kg
  ...
  CASE ROB_NOT_LOAD_VAL:
  ! Not possible to do load identification of payload
  ! in actual robot type
  ...
ENDTEST
```

Verwendung der vordefinierten Konstanten `ROB_LOAD_VAL`, `ROB_LM1_LOAD_VAL` und `ROB_NOT_LOAD_VAL` des Datentyps `paridvalidnum`.

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `paridvalidnum` sind vordefiniert und werden beim Prüfen der zurückgegebenen Werte der Funktion `ParIdRobValid` verwendet.

Wert	Symbolische Konstante	Kommentar
10	ROB_LOAD_VAL	Gültiger Robotertyp für die aktuelle Parameteridentifikation
11	ROB_NOT_LOAD_VAL	Gültiger Robotertyp für die aktuelle Parameteridentifikation
12	ROB_LM1_LOAD_VAL	Gültiger Robotertyp IRB 6400FHDfür die aktuelle Parameteridentifikation, wenn aktuelle Last < 200 kg

Eigenschaften

`paridvalidnum` ist ein Alias-Datentyp für `num` und übernimmt dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.52 paridvalidnum - Ergebnis von ParIdRobValid

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Vordefiniertes Programm Load Identify	*Bedienungsanleitung - IRC5 mit FlexPendant*
Gültiger Robotertyp	*ParIdRobValid - Gültiger Robotertyp für Parameteridentifikation auf Seite 1424*
Gültige Roboterposition	*ParIdPosValid - Gültige Roboterposition für Parameteridentifikation auf Seite 1421*
Lastidentifikation mit komplettem Beispiel	*LoadId - Lastidentifikation der Werkzeug- oder Nutzlast auf Seite 351*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.53 pathrecid - Bezeichner der Bahnaufzeichnung

Verwendung

Mit `pathrecid` wird ein Stopppunkt für die Bahnaufzeichnung angegeben.

Beschreibung

Die Bahnaufzeichnung ist eine Systemfunktion für die Aufzeichnung der vom Roboter beschriebenen Bahn. Daten des Typs `pathrecid` können über die Instruktion `PathRecStart` einer bestimmten Bahnposition zugeordnet werden. Der Anwender kann der Aufzeichnung über die Instruktion `PathRecMoveBwd` den Befehl erteilen, sich zurück zum Bahnbezeichner zu bewegen.

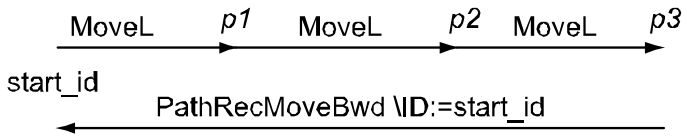
Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `pathrecid`:

Beispiel 1:

```
VAR pathrecid start_id;
CONST robtarget p1 := [...];
CONST robtarget p2 := [...];
CONST robtarget p3 := [...];

PathRecStart start_id;
MoveL p1, vmax, z50, tool1;
MoveL p2, vmax, z50, tool1
MoveL p3, vmax, z50, tool1;
IF(PathRecValidBwd (\ID := start_id)) THEN
  StorePath;
  PathRecMoveBwd \ID:=start_id;
  ...
ENDIF
```



pathrecid_Ex

Das vorhergehende Beispiel startet die Bahnaufzeichnung und der Startpunkt wird mit dem Bahnbezeichner `start_id` versehen. Danach wird der Roboter mit den herkömmlichen Bewegungsinstruktionen nach vorne und dann entsprechend der aufgezeichneten Bahn wieder zurück zur Startposition bewegt. Die Verwendung von PathRecorder-Bewegungsinstruktionen ist nur möglich, wenn die Bahnebene mit `StorePath` geändert wird.

Eigenschaften

`pathrecid` ist ein Datentyp ohne einen Wert (Non-Value).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.53 pathrecid - Bezeichner der Bahnaufzeichnung

Path Recovery

Fortsetzung

Weitere Informationen

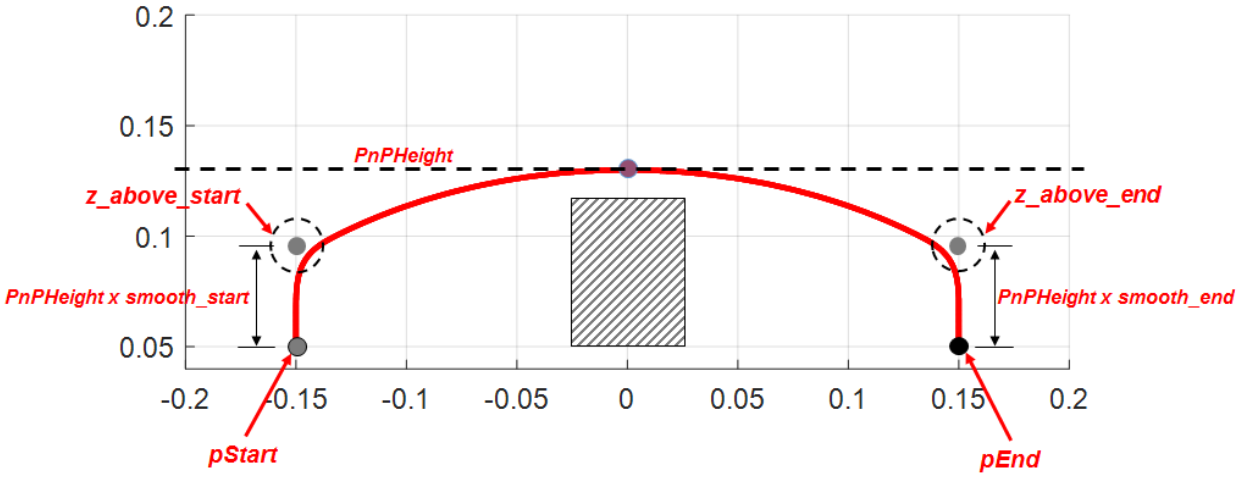
Information über	Siehe
Starten/Stoppen der Bahnaufzeichnung	*PathRecStart - Starten der Bahnaufzeichnung auf Seite 528* *PathRecStop - Stoppen der Bahnaufzeichnung auf Seite 531*
Prüfung auf gültige aufgezeichnete Bahn	*PathRecValidBwd - Prüfen, ob eine gültige Rückwärtsbahn aufgezeichnet wurde auf Seite 1431* *PathRecValidFwd - Prüfen, ob eine gültige Vorwärtsbahn aufgezeichnet wurde auf Seite 1434*
Abspielen der Bahnaufzeichnung rückwärts	*PathRecMoveBwd - Bahnaufzeichnung rückwärts ablaufen lassen auf Seite 517*
Abspielen der Bahnaufzeichnung vorwärts	*PathRecMoveFwd - Bahnaufzeichnung vorwärts ablaufen lassen auf Seite 525*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.54 pnpdata – Konfiguration von Aufnahme- und Platzierbahnen

Verwendung

`pnpdata` wird zur Konfiguration von Aufnahme- und Platzierbahnen verwendet.



xx1700001194

Beschreibung

Daten des Typs `pnpdata` beschreiben die Höhe der vertikalen Bewegung als prozentualen Anteil der gesamten Bahnhöhe und der Zonen der erzeugten Zonenbahnen.

Grundlegende Beispiele

Siehe *MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn auf Seite 475*.

Komponenten

Der Datentyp hat die folgenden Komponenten:

`smooth_start`

Datentyp: `num`

Prozentualer Anteil von `PnPHeight`, zur Beschreibung der Höhe der vertikalen Bewegung über dem Startpunkt.

Ein niedrigerer Wert kann die Zykluszeit erheblich verringern.

Der Standardwert ist 100.

`smooth_end`

Datentyp: `num`

Prozentualer Anteil von `PnPHeight`, zur Beschreibung der Höhe der vertikalen Bewegung über dem Endpunkt.

Ein niedrigerer Wert kann die Zykluszeit erheblich verringern.

Der Standardwert ist 100.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.54 pnpdata – Konfiguration von Aufnahme- und Platzierbahnen

SCARA robots

Fortsetzung

`z_above_start`

Datentyp: `zonedata`

Zonendaten für die Bewegung. `z_above_start` **beschreibt die Größe der erzeugten Zonenbahn am Ende der vertikalen Bewegung über dem Startpunkt.**

Der Standardwert ist `z100`**.**

`z_above_end`

Datentyp: `zonedata`

Zonendaten für die Bewegung. `z_above_end` **beschreibt die Größe der erzeugten Zonenbahn zu Beginn der vertikalen Bewegung über dem Endpunkt.**

Der Standardwert ist `z100`**.**

Struktur

```
<dataobject of pnpdata>
  <smooth_start of num>
  <smooth_end of num>
  <z_above_start of zonedata>
  <z_above_end of zonedata>
```

Weitere Informationen

Information über	Siehe
Bewegung des Roboters entlang einer Aufnahme- und Platzierbahn	*MovePnP – Bewegt den Roboter entlang einer Aufnahme- und Platzierbahn auf Seite 475*
	Technisches Referenzhandbuch - RAPID Overview

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.55 pos – Positionen (nur X, Y und Z)

Verwendung

`pos` wird für Positionen (nur X, Y und Z) verwendet.

Der Datentyp `robtarget` wird für die Position des Roboters einschließlich der Werkzeugorientierung und der Achsenkonfiguration verwendet.

Beschreibung

Daten des Typs `pos` beschreiben die Koordinaten einer Position: X, Y und Z.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `pos`:

Beispiel 1:

```
VAR pos pos1;
...
pos1 := [500, 0, 940];
```

Der Position `pos1` wird folgender Wert zugewiesen: X = 500 mm, Y = 0 mm, Z = 940 mm.

Beispiel 2:

```
pos1.x := pos1.x + 50;
```

Die Position `pos1` wird um `50` mm in X-Richtung verschoben.

Komponenten

Der Datentyp `pos` hat die folgenden Komponenten:

x

Datentyp: `num`

X-Wert der Position.

y

Datentyp: `num`

Y-Wert der Position.

z

Datentyp: `num`

Z-Wert der Position.

Struktur

```
< dataobject of pos >
  < x of num >
  < y of num >
  < z of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.55 pos – Positionen (nur X, Y und Z)
RobotWare - OS
Fortsetzung

Weitere Informationen

Information über	Siehe
Positionsoperationen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Expressions*
Roboterposition und Orientierung	*robtarget - Positionsdaten auf Seite 1778*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.56 pose – Koordinatentransformationen

Verwendung

`pose` wird für den Wechsel aus einem Koordinatensystem in ein anderes verwendet.

Beschreibung

Daten des Typs `pose` beschreiben, wie ein Koordinatensystem verschoben und um ein anderes Koordinatensystem gedreht wird. Die Daten können z. B. angeben, wie das Werkzeug-Koordinatensystem in Relation zum Handgelenk-Koordinatensystem positioniert und orientiert ist.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `pose`:

```
VAR pose frame1;
...
frame1.trans := [50, 0, 40];
frame1.rot := [1, 0, 0, 0];
```

Der Koordinatentransformation `frame1` wird ein Wert zugewiesen, der einer Verschiebung der Position entspricht, wobei X = `50` mm, Y = `0` mm, Z = `40` mm ist. Es findet jedoch keine Drehung statt.

Komponenten

Der Datentyp hat die folgenden Komponenten:

`trans`

translation

Datentyp: `pos`

Die Positionsverschiebung (x, y und z) des Koordinatensystems.

`rot`

rotation

Datentyp: `orient`

Die Drehung des Koordinatensystems.

Struktur

```
< dataobject of pose >
  < trans of pos >
  < rot of orient >
```

Weitere Informationen

Information über	Siehe
Was ist ein Quaternion?	*orient – Ausrichtung auf Seite 1744*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.57 processtimes- Prozesszeiten

Verwendung

`processtimes` wird verwendet, um die Laufzeiten für alle Statusüberwachungsphasen des CAP zu definieren, außer für die Phase MAIN, die durch die Roboterbewegung definiert ist (siehe Abschnitt *Supervision* in *Application manual - Continuous Application Platform*).

`processtimes` ist eine Komponente von `capdata` und definiert die Zeitüberschreitungen für die folgenden Statusüberwachungsphasen des CAP:

- PRE_START
- POST1
- POST2

Die angegebene Zeitüberschreitung muss größer sein als Null, wenn die Überwachung während der entsprechenden Statusüberwachungsphase des CAP verwendet werden soll (siehe Abschnitt *Supervision and process phases* in *Application manual - Continuous Application Platform*).

Komponenten

pre

Datentyp: `num`

Definiert die Dauer der Phase PRE_START in Sekunden. Während dieser Zeit müssen alle für diese Phase definierten Bedingungen erfüllt sein.

post1

Datentyp: `num`

Definiert die Dauer der Phase POST1 in Sekunden. Während dieser Zeit müssen alle für diese Phase definierten Bedingungen erfüllt sein.

post2

Datentyp: `num`

Definiert die Dauer der Phase POST2 in Sekunden. Während dieser Zeit müssen alle für diese Phase definierten Bedingungen erfüllt sein.

Syntax

```
< data object of processtimes >
  < pre of num >
  < post1 of num >
  < post2 of num >
```

Weitere Informationen

	Beschrieben in:
`capdata` Datentyp	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.58 progdisp - Programmverschiebung

Verwendung

`progdisp` wird zum Speichern der aktuellen Programmverschiebung des Roboters und der externen Achsen verwendet.

Dieser Datentyp muss normalerweise nicht verwendet werden, weil die Daten mit den Instruktionen `PDispSet`, `PDispOn`, `PDispOff`, `EOffsSet`, `EOffsOn` und `EOffsOff` gesetzt werden. Dieser Datentyp dient lediglich dem temporären Speichern des aktuellen Werts für spätere Verwendung.

Beschreibung

Auf die aktuellen Werte der Programmverschiebung können Sie mit der Systemvariable `C_PROGDISP` zugreifen.

Weitere Informationen erhalten Sie unter den Instruktionen `PDispSet`, `PDispOn`, `EOffsSet` und `EOffsOn`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `progdisp`:

Beispiel 1:

```
VAR progdisp progdisp1;
...
SearchL sen1, psearch, p10, v100, tool1;
PDispOn \ExeP:=psearch, *, tool1;
EOffsOn \ExeP:=psearch, *;
...
progdisp1:=C_PROGDISP;
PDispOff;
EOffsOff;
...
PDispSet progdisp1.pdisp;
EOffsSet progdisp1.eoffs;
```

Zuerst wird eine Programmverschiebung von einer gesuchten Position aktiviert. Dann werden die aktuellen Werte der Programmverschiebung temporär in der Variable `progdisp1` gespeichert und die Programmverschiebung wird deaktiviert. Später erfolgt die erneute Aktivierung über die Instruktionen `PDispSet` und `EOffsSet`.

Vordefinierte Daten

Die Systemvariable`C_PROGDISP` beschreibt die aktuelle Programmverschiebung des Roboters und der externen Achsen und das Programm kann immer darauf zugreifen. Sie lässt sich jedoch nur über eine Reihe von Instruktionen ändern, nicht durch Zuweisung.

Die folgenden Standardwerte für Programmverschiebung werden gesetzt:

- bei Verwendung des Neustart-Modus **RAPID zurücksetzen**.
- wenn ein neues Programm oder ein neues Modul geladen wird
- bei Beginn der Programmabarbeitung vom Anfang

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.58 progdisp - Programmverschiebung
RobotWare - OS
Fortsetzung

- beim Verschieben des Programmzeigers auf `main`
- wenn der Programmzeiger auf eine Routine verschoben wird
- wenn der Programmzeiger so verschoben wird, dass die Ausführungsreihenfolge verloren geht.

```
VAR progdisp C_PROGDISP :=
  [ [[ 0, 0, 0], [1, 0, 0, 0]],-> posedata
  [ 0, 0, 0, 0, 0, 0]];-> extjointdata
```

Komponenten

`pdisp`

program displacement

Datentyp: `pose`

Die Programmverschiebung für den Roboter, ausgedrückt mit einer Translation und einer Orientierung. Die Translation wird in mm angegeben.

`eoffs`

external offset

Datentyp: `extjoint`

Der Offset für jede externe Achse. Für eine lineare Achse wird der Wert in mm, für eine Rotationsachse in Grad angegeben.

Struktur

```
< dataobject of progdisp >
  < pdisp of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
  < eoffs of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
```

Weitere Informationen

Information über	Siehe
Instruktionen für Programmverschiebungen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion settings*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - Coordinate systems*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.59 rawbytes – Unverarbeitete Daten

Verwendung

`rawbytes` wird als allgemeiner Datencontainer verwendet. Es kann für die Kommunikation mit E/A-Geräten eingesetzt werden.

Beschreibung

`rawbytes`-Daten können mithilfe von Unterstützungsinstruktionen/-funktionen mit einem beliebigen Datentyp - `num`, `byte`, `string` - gefüllt werden. In jeder Variable vom Typ `rawbytes` wird außerdem die aktuelle Länge gültiger Bytes vom System gespeichert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `rawbytes`:

Beispiel 1:

```
VAR rawbytes raw_data;
VAR num integer := 8;
VAR num float := 13.4;

ClearRawBytes raw_data;
PackRawBytes integer, raw_data, 1 \IntX := INT;
PackRawBytes float, raw_data, (RawBytesLen(raw_data)+1) \Float4;
```

In diesem Beispiel wird die Variable `raw_data` des Typs `rawbytes` zunächst gelöscht, d. h., alle Byte werden auf 0 gesetzt (wie Standard bei Deklaration). In die ersten 2 Byte wird der Wert von `integer` übernommen und in die nächsten 4 Byte der Wert von `float`.

Einschränkungen

Eine `rawbytes`-Variable kann zwischen 0 und 1024 Byte enthalten.

Struktur

`rawbytes` ist ein Datentyp ohne einen Wert (Non-Value).

Bei der Deklaration der `rawbytes`-Variable werden alle Bytes in `rawbytes` auf 0 gesetzt und die aktuelle Länge gültiger Bytes in der Variable wird auf 0 gesetzt.

Weitere Informationen

Information über	Siehe
Abrufen der Länge von `rawbytes`-Daten	*RawBytesLen - Abrufen der Länge von rawbytes-Daten auf Seite 1454*
Löschen des Inhalts von `rawbytes`-Daten	*ClearRawBytes - Löschen des Inhalts von rawbytes-Daten auf Seite 146*
Kopieren des Inhalts von `rawbytes`-Daten	*CopyRawBytes - Kopieren des Inhalts von rawbytes-Daten auf Seite 171*
Packen des DeviceNet-Headers in `rawbytes`-Daten	*PackDNHeader - Packen des DeviceNet-Headers in rawbytes-Daten auf Seite 499*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Packen von Daten in `rawbytes`-Daten	*PackRawBytes - Packen von Daten in rawbytes-Daten auf Seite 502*
Schreiben von `rawbytes`-Daten	*WriteRawBytes - Schreiben von rawbytes-Daten auf Seite 1127*
Lesen von `rawbytes`-Daten	*ReadRawBytes - Lesen der rawbytes-Daten auf Seite 593*
Entpacken von Daten aus `rawbytes`-Daten	*UnpackRawBytes - Entpacken von Daten aus rawbytes-Daten auf Seite 1033*
Datei- und E/A-Gerät-Handhabung	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.60 restartblkdata - Blockdaten für Neustart

Verwendung

`restartblkdata` werden verwendet, um das Verhalten beim Neustart des Prozesses zu definieren.

`restartblkdata` ist eine Komponente der `capdata` und definiert Folgendes für einen CAP-Prozess beim Neustart, wenn:

- Der Roboter die stationäre Pendelbewegung während des Prozessneustarts ausführen/blockieren soll (`weave_start`).
- Der Neustart der Roboterbewegung relativ zum Prozess verzögert oder nicht verzögert werden soll (`motion_delay`).
- Die PRE-Phase und diePRE_START-Phase ausgeführt/blockiert werden sollen (`pre_phase`).
- Während des Prozessstarts eine von der Hauptgeschwindigkeit verschiedene Geschwindigkeit verwendet oder nicht verwendet werden soll (`startspeed_phase`).
- Die Phasen START_POST1 und POST1 ausgeführt/blockiert werden sollen (`post1_phase`).
- Die Phasen START_POST2 und POST2 ausgeführt/blockiert werden sollen (`post2_phase`).

Komponenten

weave_start

Datentyp: `bool`

Wert	Beschreibung
FALSE	Stationäre Pendelbewegungen beim Neustart, bis der Prozess begonnen hat
TRUE	Keine stationären Pendelbewegungen beim Neustart, bis der Prozess begonnen hat

motion_delay

Datentyp: `bool`

Wert	Beschreibung
FALSE	Verzögerung der Roboterbewegung beim Neustart, nachdem der Prozess begonnen hat
TRUE	Keine Verzögerung der Roboterbewegung beim Neustart, nachdem der Prozess begonnen hat

pre_phase

Datentyp: `bool`

Wert	Beschreibung
FALSE	Ausführen der Phasen PRE und PRE_START beim Neustart
TRUE	Phasen PRE und PRE_START beim Neustart NICHT ausführen

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

startspeed_phase

Datentyp: `bool`

Wert	Beschreibung
FALSE	Bewegt den Roboter am Anfang eines Neustarts mit Startgeschwindigkeit
TRUE	Roboter am Anfang eines Neustarts NICHT mit Startgeschwindigkeit bewegen, sondern direkt die Hauptgeschwindigkeit verwenden

post1_phase

Datentyp: `bool`

Wert	Beschreibung
FALSE	Ausführen der Phasen START_POST1 und POST1 beim Neustart
TRUE	Phasen START_POST1 und POST1 beim Neustart NICHT ausführen

post2_phase

Datentyp: `bool`

Wert	Beschreibung
FALSE	Ausführen der Phasen START_POST2 und POST2 beim Neustart
TRUE	Phasen START_POST2 und POST2 beim Neustart NICHT ausführen

Syntax

```
< data object of restartblkdata >
  < weave_start of bool >
  < motion_delay of bool >
  < pre_phase of bool >
  < startspeed_phase of bool >
  < post1_phase of bool >
  < post2_phase of bool >
```

Weitere Informationen

	Beschrieben in:
`capdata` Datentyp	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.61 restartdata - Neustartdaten für Trigg-Signale

Verwendung

`restartdata` gibt die Vor- und Nachwerte ausgewählter E/A-Signale (Prozessignale) an der Stoppsequenz der Roboterbewegungen an. Die E/A-Signale zur Überwachung werden in der Instruktion `TriggStopProc` angegeben.

`TriggStopProc` und `restartdata` dienen dem Neustart nach einem Programmstopp (STOP) oder Not-Halt (QSTOP) der eigenen Prozessinstruktionen, die in RAPID (NOSTEPIN-Routinen) definiert sind.

Beschreibung

`restartdata` gibt folgende Daten an, nachdem die Abarbeitung des Programms unterbrochen wurde:

- gültige Neustartdaten
- Angabe, ob der Roboter auf der Bahn angehalten wurde
- Vorwert des E/A-Signals
- Nachwert des E/A-Signals
- Anzahl der Flanken zwischen Vor- und Nachzeit des Schattensignals für den laufenden Prozess

Definition

Die Tabelle zeigt die Definition des Zeitpunkts für das Lesen der Vor- und Nachwerte für die I/O-Signale.

Stopppunkt-Typ	Lesen der Zeit für den Vorwert des E/A-Signals	Lesen der Zeit für den Nachwert des E/A-Signals
STOP auf der Bahn	Bei Stillstand aller Roboterachsen	Ca. 400 ms nach der Vorzeit
QSTOP Bahnabweichung	So schnell wie möglich	Ca. 400 ms nach der Vorzeit

Komponenten

`restartstop`

valid restartdata after stop

Datentyp: `bool`

TRUE = Letzten STOP oder QSTOP anzeigen

FALSE = Ungültige Neustartdaten. Alle E/A-Signale werden auf -1 gesetzt.

`stoponpath`

stop on path

Datentyp: `bool`

TRUE = Der Roboter wird auf der Bahn angehalten (STOP).

FALSE = Der Roboter wird angehalten, jedoch nicht auf der Bahn (QSTOP)

`predo1val`

pre do1 value

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `dionum`

Der Vorwert des digitalen Signals "do1" wird im Argument `DO1` in der Instruktion `TriggStopProc` angegeben.

`postdo1val`

post do1 value

Datentyp: `dionum`

Der Nachwert des digitalen Signals "do1" wird im Argument `DO1` in der Instruktion `TriggStopProc` angegeben.

`prego1val`

pre go1 value

Datentyp: `num`

Der Vorwert des digitalen Gruppensignals "go1" wird im Argument `GO1` in der Instruktion `TriggStopProc` angegeben.

`postgo1val`

post go1 value

Datentyp: `num`

Der Nachwert des digitalen Gruppensignals "go1" wird im Argument `GO1` in der Instruktion `TriggStopProc` angegeben.

`prego2val`

pre go2 value

Datentyp: `num`

Der Vorwert des digitalen Gruppensignals "go2" wird im Argument `GO2` in der Instruktion `TriggStopProc` angegeben.

`postgo2val`

post go2 value

Datentyp: `num`

Der Nachwert des digitalen Gruppensignals "go2" wird im Argument `GO2` in der Instruktion `TriggStopProc` angegeben.

`prego3val`

pre go3 value

Datentyp: `num`

Der Vorwert des digitalen Gruppensignals "go3" wird im Argument `GO3` in der Instruktion `TriggStopProc` angegeben.

`postgo3val`

post go3 value

Datentyp: `num`

Der Nachwert des digitalen Gruppensignals "go3" wird im Argument `GO3` in der Instruktion `TriggStopProc` angegeben.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.61 restartdata - Neustartdaten für Trigg-Signale
RobotWare - OS
Fortsetzung

`prego4val`

pre go4 value

Datentyp: `num`

Der Vorwert des digitalen Gruppensignals "go4" wird im Argument `GO4` in der Instruktion `TriggStopProc` angegeben.

`postgo4val`

post go4 value

Datentyp: `num`

Der Nachwert des digitalen Gruppensignals "go4" wird im Argument `GO4` in der Instruktion `TriggStopProc` angegeben.

`preshadowval`

pre shadow value

Datentyp: `dionum`

Der Vorwert des digitalen Signals "Schatten" wird im Argument `ShadowDO` in der Instruktion `TriggStopProc` angegeben.

`shadowflanks`

number of shadow flanks

Datentyp: `num`

Die Anzahl der Werteübergänge (Flanken) des digitalen Schattensignals zwischen der Vorzeit und der Nachzeit. Das Schattensignal wird im Argument `ShadowDO` in der Instruktion `TriggStopProc.` angegeben.

`postshadowval`

post shadow value

Datentyp: `dionum`

Der Nachwert des digitalen Signals "Schatten" wird im Argument `ShadowDO` in der Instruktion `TriggStopProc` angegeben.

Struktur

```
< dataobject of restartdata >
  < restartstop of bool >
  < stoponpath of bool >
  < predo1val of dionum >
  < postdo1val of dionum >
  < prego1val of num >
  < postgo1val of num >
  < prego2val of num >
  < postgo2val of num >
  < prego3val of num >
  < postgo3val of num >
  < prego4val of num >
  < postgo4val of num >
  < preshadowval of dionum >
  < shadowflanks of dionum >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
< postshadowval of dionum >
```

Weitere Informationen

Information über	Siehe
Vordefinierte Prozessinstruktionen	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904*
Einrichtung der Wiedergabe der Neustartdaten	*TriggStopProc - Generieren von Neustartdaten für Trigg-Signale bei einem Stopp auf Seite 994*
Rückwärtsbewegung auf der Bahn	*StepBwdPath - Rückwärtsbewegung um einen Schritt auf der Bahn auf Seite 821*
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.62 rmqheader - RAPID Message Queue-Meldungsheader

Verwendung

`rmqheader` (*RAPID Message Queue Header*) wird zum Lesen der Datenstruktur der Daten in einer Meldung vom Typ `rmqmessage` verwendet.

Beschreibung

Der Header-Typ einer `rmqmessage` von einem Datentyp ohne einen Wert, die in den `rmqheader` von einem Wertdatentyp konvertiert wurde.

Beispiele

Grundlegende Beispiele für den Datentyp `rmqheader` sind unten aufgeführt.

Beispiel 1:

```
VAR rmqmessage message;
VAR rmqheader header;
...
RMQGetMessage message;
RMQGetMsgHeader message \Header:=header;
```

Kopiert und konvertiert die `rmqheader`-Informationen aus einer `rmqmessage`.

Komponenten

`datatype`

Datentyp: `string`

Der Name des verwendeten Datentyps, z. B. von `num`, `string` oder eines anderen Wertdatentyps.

`ndim`

Number of Dimensions

Datentyp: `num`

Anzahl der Datenfelddimensionen.

`dim1`

Size of first dimension

Datentyp: `num`

Die Größe der ersten Dimension. 0 wird nicht verwendet.

`dim2`

Size of second dimension

Datentyp: `num`

Die Größe der zweiten Dimension. 0 wird nicht verwendet.

`dim3`

Size of third dimension

Datentyp: `num`

Die Größe der dritten Dimension. 0 wird nicht verwendet.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Struktur

```
<dataobject of rmqheader>
  <datatype of string>
  <ndim of num>
  <dim1 of num>
  <dim2 of num>
  <dim3 of num>
```

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
`RMQ Message`	*rmqmessage - RAPID Message Queue-Meldung auf Seite 1774*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.63 rmqmessage - RAPID Message Queue-Meldung

Verwendung

`rmqmessage` (*RAPID Message Queue Message*) wird für die temporäre Speicherung von Kommunikationsdaten verwendet.

Beschreibung

Der Datentyp `rmqmessage` ist die Meldung, in der Daten bei der Kommunikation zwischen verschiedenen RAPID-Tasks oder Robot Application Builder-Clients mit RMQ-Funktionalität gespeichert werden. Sie enthält Informationen über den versendeten Datentyp, die Dimensionen der Daten, die Identität des Senders und die eigentlichen Daten.

Eine `rmqmessage` ist ein umfangreicher Datentyp (ca. 3000 Byte), und es wird empfohlen, die Variable wiederzuverwenden, um RAPID-Speicher zu sparen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `rmqmessage`:

Beispiel 1:

```
VAR rmqmessage rmqmessage1;
VAR string myrecdata;
...
RMQGetMsgData rmqmessage1, myrecdata;
```

Die Variable `rmqmessage1` wird in einem RMQ (RAPID Message Queue)-Befehl definiert und verwendet. In diesem Beispiel wird der Datenteil der `rmqmessage1` in die Variable `myrecdata` kopiert.

Eigenschaften

`rmqmessage` ist ein Datentyp ohne einen Wert (Non-Value) und kann nicht in wertorientierten Operationen verwendet werden.

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
`RMQ Header`	*rmqheader - RAPID Message Queue-Meldungsheader auf Seite 1772*
Extrahieren der Headerdaten aus einer `rmqmessage`	*RMQGetMsgHeader - Abrufen der Headerinformationen aus einer RMQ-Meldung auf Seite 637*
Anfordern und Aktivieren von Interrupts für einen bestimmten Datentyp	*IRMQMessage - Anfordern von RMQ-Interrupts für einen Datentyp auf Seite 303*
Abrufen der ersten Meldung aus einer RAPID Message Queue	*RMQGetMessage - Abrufen einer RMQ-Meldung auf Seite 631*
Senden von Daten an die Warteschlange einer RAPID-Task oder eines Robot Application Builder-Clients und Warten auf eine Antwort vom Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Extrahieren der Daten aus einer `rmqmessage`	*RMQGetMsgData - Abrufen der Daten aus einer RMQ-Meldung auf Seite 634*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.64 rmqslot - Identitätsnummer eines RMQ-Clients

Verwendung

`rmqslot` (*RAPID Message Queue Slot*) wird bei der Kommunikation mit einer RMQ oder einem Robot Application Builder-Client verwendet.

Beschreibung

Der `rmqslot` ist eine Identitätsnummer einer für eine RAPID-Task konfigurierten RAPID Message Queue oder eines Robot Application Builder-Clients.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `rmqslot`:

Beispiel 1:

```
VAR rmqslot rmqslot1;
RMQFindSlot rmqslot1, "RMQ_T_ROB1";
...
```

Die Variable `rmqslot1` wird in der Instruktion `RMQFindSlot` definiert und verwendet, um die Identitätsnummer der für die RAPID-Task `RMQ_T_ROB1` konfigurierten RAPID Message Queue `T_ROB1` abzurufen.

Eigenschaften

`rmqslot` ist ein Datentyp ohne einen Wert (Non-Value) und kann nicht in wertorientierten Operationen verwendet werden.

Weitere Informationen

Information über	Siehe
Beschreibung der RAPID Message Queue-Funktionalität	*Anwendungshandbuch - Steuerungssoftware IRC5*, Abschnitt *RAPID Message Queue*.
Suchen der Identitätsnummer einer RAPID-Message Queue-Task oder eines Robot Application Builder-Clients	*RMQFindSlot - Suchen einer Slot-Identität anhand des Slot-Namens auf Seite 629*
Senden von Daten an die Warteschlange einer RAPID-Task oder eines Robot Application Builder-Clients	*RMQSendMessage - Senden einer RMQ-Datenmeldung auf Seite 643*
Senden von Daten an einen Client und Warten auf eine Antwort von dem Client	*RMQSendWait - Senden einer RMQ-Datenmeldung und Warten auf eine Antwort auf Seite 647*
Abrufen des Slot-Namens aus einer angegebenen Slot-Identität	*RMQGetSlotName - Abrufen des Namens eines RMQ-Clients auf Seite 1479*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.65 robjoint – Position der Roboterachsen

Verwendung

`robjoint` wird zur Definition der Achsenposition in Grad der Roboterachsen verwendet.

Beschreibung

Daten des Typs `robjoint` werden verwendet, um Achsenpositionen in Grad der Roboterachsen 1 bis 6 zu speichern. Die Achsenposition wird als Drehung in Grad für die jeweilige Achse (den Arm) in positiver oder negativer Richtung ab der Achsenkalibrierposition definiert.

Komponenten

`rax_1`

robot axis 1

Datentyp: `num`

Die Position von Roboterachse 1 in Grad von der Kalibrierposition aus.

`...`

`rax_6`

robot axis 6

Datentyp: `num`

Die Position von Roboterachse 6 in Grad von der Kalibrierposition aus.

Struktur

```
< dataobject of robjoint >
  < rax_1 of num >
  < rax_2 of num >
  < rax_3 of num >
  < rax_4 of num >
  < rax_5 of num >
  < rax_6 of num >
```

Weitere Informationen

Information über	Siehe
Daten über die Achsenposition	*jointtarget – Achsenpositionsdaten auf Seite 1720*
Bewegen zur Achsenposition	*MoveAbsJ - Bewegen des Roboters an eine absolute Achsenposition auf Seite 382*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.66 robtarget - Positionsdaten

Verwendung

`robtarget` (*robot target*) wird zur Definition der Position des Roboters und der zusätzlichen Achsen verwendet.

Beschreibung

Anhand der Positionsdaten wird in den Bewegungsinstruktionen die Position definiert, an die sich der Roboter und zusätzliche Achsen bewegen sollen.

Da der Roboter dieselbe Position mittels mehrerer Möglichkeiten erreichen kann, wird auch die Achsenkonfiguration angegeben. Damit werden die Achsenwerte definiert, falls diese nicht eindeutig sind, z. B.:

- wenn sich der Roboter in einer Vorwärts- oder Rückwärts-Position befindet,
- wenn Achse 4 nach unten oder oben weist,
- wenn Achse 6 eine negative oder positive Drehung aufweist.

WARNUNG

Die Position wird mit dem Koordinatensystem des Werkobjekts einschließlich jeglicher Programmverschiebung definiert. Wenn die Position mit einem Werkobjekt programmiert wurde, das von dem in der Instruktion abweicht, bewegt sich der Roboter nicht wie erwartet. Stellen Sie sicher, dass Sie dasselbe Werkobjekt wie beim Programmieren der Bewegungsinstruktionen verwenden. Ein Fehler kann zu Verletzungen oder Beschädigung des Roboters oder anderer Geräte führen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `robtarget`:

Beispiel 1:

```
CONST robtarget p15 := [ [600, 500, 225.3], [1, 0, 0, 0], [1, 1,
     0, 0], [ 11, 12.3, 9E9, 9E9, 9E9, 9E9] ];
```

Eine Position `p15` wird wie folgt definiert:

- Die Position des Roboters: `x` = 600, `y` = 500, und `z` = 225,3 mm im Objekt-Koordinatensystem.
- Die Orientierung des Werkzeugs hat die gleiche Richtung wie das Objekt-Koordinatensystem.
- Die Achsenkonfiguration des Roboters: Achse 1 und 4 in Position 90-180°, Achse 6 in Position 0-90°.
- Die Position der zusätzlichen logischen Achsen a (11) und b (12.3), ausgedrückt in Grad oder mm (abhängig vom Typ der Achse). Achsen c bis f sind undefiniert (9E9).

Beispiel 2:

```
VAR robtarget p20;
...
p20 := CRobT(\Tool:=tool\wobj:=wobjØ);
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
p20 := Offs(p20,10,0,0);
```

Die Position `p20` wird durch Aufruf der Funktion `CRobT` auf die aktuelle Position des Roboters gesetzt. Die Position wird dann um 10 mm in x-Richtung verschoben.

Komponenten

`trans`

translation

Datentyp: `pos`

Die Position (x, y und z) des Werkzeugarbeitspunkts in mm ausgedrückt.

Die Position wird in Relation zum aktuellen Objekt-Koordinatensystem einschließlich Programmverschiebung angegeben. Wenn kein Werkobjekt angegeben ist, ist dies das Welt-Koordinatensystem.

`rot`

rotation

Datentyp: `orient`

Die Orientierung des Werkzeugs als Quaternion (q1, q2, q3 und q4) ausgedrückt.

Die Orientierung wird in Relation zum aktuellen Objekt-Koordinatensystem, einschließlich Programmverschiebung, angegeben. Wenn kein Werkobjekt angegeben ist, ist dies das Welt-Koordinatensystem.

`robconf`

robot configuration

Datentyp: `confdata`

Die Achsenkonfiguration des Roboters (`cf1`, `cf4`, `cf6` und `cfx`). Für Industrieroboter wird dies in Form der aktuellen Viertelumdrehung von Achse 1, Achse 4 und Achse 6 definiert. Die Bedeutung der Komponente `cfx` ist abhängig vom Typ des Roboters. Für weitere Informationen siehe den Datentyp *confdata – Konfigurationsdaten des Roboters auf Seite 1677*.

`extax`

external axes

Datentyp: `extjoint`

Die Position der zusätzlichen Achsen.

Die Position wird folgendermaßen für jede einzelne Achse definiert (`eax_a`, `eax_b ... eax_f`):

- Für Rotationsachsen wird die Position als Drehung in Grad ab der Kalibrierposition definiert.
- Für lineare Achsen wird die Position als Abstand in mm ab der Kalibrierposition definiert.

Zusätzliche Achsen `eax_a ...` sind logische Achsen. Die Beziehung zwischen der logischen Achsennummer und der physischen Achsennummer wird in den Systemparametern definiert.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.66 robtarget - Positionsdaten
RobotWare - OS
Fortsetzung

Der Wert 9E9 wird für Achsen definiert, die nicht verbunden sind. Wenn sich die in den Positionsdaten definierten Achsen von den Achsen unterscheiden, die tatsächlich bei der Programmabarbeitung verbunden sind, gilt Folgendes:

- Wenn die Position nicht in den Positionsdaten definiert ist (Wert 9E9), wird der Wert ignoriert, falls die Achse verbunden und nicht aktiviert ist. Wenn aber die Achse aktiviert ist, ergibt das einen Fehler.
- Wenn die Position in den Positionsdaten definiert ist, obwohl die Achse nicht verbunden ist, wird der Wert ignoriert.

Es wird keine Bewegung durchgeführt, aber für eine Achse mit gültigen Positionsdaten wird kein Fehler ausgegeben, wenn die Achse nicht aktiviert wird.

Wenn eine zusätzlich Achse im unabhängigen Modus arbeitet und vom Roboter und seinen zusätzlichen Achsen eine neue Bewegung ausgeführt werden soll, darf als Positionsdaten für die zusätzliche Achse im unabhängigen Modus nicht 9E9 verwendet werden. Die Daten müssen ein beliebiger Wert sein, der nicht vom System verwendet wird.

Struktur

```
< dataobject of robtarget >
  < trans of pos >
    < x of num >
    < y of num >
    < z of num >
  < rot of orient >
    < q1 of num >
    < q2 of num >
    < q3 of num >
    < q4 of num >
  < robconf of confdata >
    < cf1 of num >
    < cf4 of num >
    < cf6 of num >
    < cfx of num >
  < extax of extjoint >
    < eax_a of num >
    < eax_b of num >
    < eax_c of num >
    < eax_d of num >
    < eax_e of num >
    < eax_f of num >
```

Weitere Informationen

Information über	Siehe
Bewegungsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Motion*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Coordinate Systems*
Handhabung von Konfigurationsdaten	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Robot configuration*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Konfiguration der zusätzlichen Achsen	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*
Was ist ein Quaternion?	*orient – Ausrichtung auf Seite 1744*
`confdata`	*confdata – Konfigurationsdaten des Roboters auf Seite 1677*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.67 sensor - Externer Gerätedeskriptor

Verwendung

`sensor` ist ein Deskriptor für die Kopplung externer Geräte.

Beschreibung

Der Deskriptor eines Geräts auf der Ebene RAPID ist im Aufzeichnungsdatentyp `sensor` enthalten. Er beinhaltet Informationen zum Sensorgerät wie ID, Fehlercode und Kommunikationsstatus des Sensors.

Komponenten

`id`

Datentyp: `num`

Der interne Bezeichner des Geräts, das beim ersten Betrieb mit dem Gerät von Ebene RAPID eingestellt wird. (Noch nicht implementiert).

`error`

Datentyp: `num`

Der Parameter `error` wird eingestellt, wenn der Parameter `state` auf `STATE_ERROR` gestellt wird. Wenn `state` von `STATE_ERROR` zu `STATE_CONNECTED` wechselt, wird der Parameter `error` auf 0 gestellt.

Fehlernummer	Fehler
0	Kein Fehler.
112600	Fehler bei der Initialisierung der Kommunikationsschnittstelle.
112602	Fehler der Kommunikationsschnittstelle.

`state`

Datentyp: `sensorstate`

Gibt den aktuellen Kommunikationszustand des Geräts wieder.

Beispiele

Ein Beispiel für den Datentyp `sensor` wird unten dargestellt.

Beispiel 1:

```
PERS sensor AnyDevice;
PERS robdata DataOut := [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
PERS sensdata DataIn :=
     ["No",[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]];
VAR num SampleRate:=64;
...
! Setup Interface Procedure
PROC RRI_Open()
  SiConnect AnyDevice;
  ! Send and receive data cyclic with 64 ms rate
  SiGetCyclic AnyDevice, DataIn, SampleRate;
  SiSetCyclic AnyDevice, DataOut, SampleRate;
ENDPROC
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beim Aufruf der Routine `RRI_Open` wird zunächst eine Verbindung zum Gerät `AnyDevice` geöffnet. Dann beginnt eine zyklische Übertragung mit der Frequenz `SampleRate`.

Struktur

```
<dataobject of sensor>
  <id of num>
  <error of num>
  <state of sensorstate>
```

Weitere Informationen

Information über	Siehe
Aufbau einer Verbindung zu einem externen System.	*SiConnect - Sensorschnittstelle verbinden auf Seite 728*.
Schließen einer Verbindung zu einem externen System.	*SiClose - Sensorschnittstelle schließen auf Seite 731*.
Datenregistrierung für die zyklische Übertragung.	*SiSetCyclic - Sensorschnittstelle zyklisch einstellen auf Seite 739*.
Für eine zyklische Datenübertragung anmelden.	*SiGetCyclic - Sensorschnittstelle zyklisch abrufen auf Seite 733*
Kommunikationszustand eines Geräts.	*sensorstate - Kommunikationszustand des Geräts auf Seite 1784*.
Robot Reference Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.68 sensorstate - Kommunikationszustand des Geräts

Verwendung

`sensorstate` wird verwendet, um den tatsächlichen Kommunikationszustand eines Geräts darzustellen.

Beschreibung

Eine Konstante `sensorstate` wird verwendet, um den tatsächlichen Kommunikationszustand eines Geräts darzustellen. Mit RAPID kann sie verwendet werden, um den Kommunikationszustand mit dem Sensor zu evaluieren.

Vordefinierte Daten

Folgende symbolische Konstanten des Datentyps `sensorstate` sind vorgegeben und können verwendet werden, um den Kommunikationszustand eines Geräts zu evaluieren.

Konstante	Wert
`STATE_ERROR`	`-1`
`STATE_UNDEFINED`	`0`
`STATE_CONNECTED`	`1`
`STATE_OPERATING`	`2`
`STATE_CLOSED`	`3`

Eigenschaften

`sensorstate` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Aufbau einer Verbindung zu einem externen System.	*SiConnect - Sensorschnittstelle verbinden auf Seite 728*.
Schließen einer Verbindung zu einem externen System.	*SiClose - Sensorschnittstelle schließen auf Seite 731*.
Datenregistrierung für die zyklische Übertragung.	*SiSetCyclic - Sensorschnittstelle zyklisch einstellen auf Seite 739*.
Für eine zyklische Datenübertragung anmelden.	*SiGetCyclic - Sensorschnittstelle zyklisch abrufen auf Seite 733*
Deskriptor für externe Geräte.	*sensor - Externer Gerätedeskriptor auf Seite 1782*.
Robot Reference Interface	*Anwendungshandbuch - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.69 sensorvardata - Mehrere variable Konfigurationsdaten für die Sensorschnittstelle

Verwendung

`sensorvardata` wird verwendet, um die benötigten Informationen für die verschiedenen Datenpunkte zu konfigurieren, die von den Befehlen `ReadVarArr` und `WriteVarArr` gehandhabt werden.

Komponenten

Der Datentyp hat die folgenden Komponenten:

`varnumber`

Datentyp: `num`

Definiert die Variablennummer, die gelesen/geschrieben werden soll.

`sensordatatype`

Datentyp: `num`

Der Datentyp, durch den der Wert dargestellt wird. Der Wert wird in den angegebenen Typ konvertiert, bevor er gesendet wird oder nachdem er über die Kommunikationsverbindung des Sensors empfangen wird.

Gültige Werte sind:

Wert	Beschreibung
0	INT16
1	UINT16
2	DOUBLE

Bei Verwendung eines Sensors von ServoRobot® über eine Ethernet-Verbindung wird immer `sensordatatype 2` (doppelt) verwendet. Für alle anderen Arten von Sensoren und Kommunikation muss `senordatatype 0` oder `1` verwendet werden.

`raw`

Datentyp: `bool`

Wenn dieses Flag True ist, erfolgt keine interne Änderung an den Daten, bevor oder nachdem sie über das Protokoll zum/vom Gerät gesendet werden. Der gesendete/empfangene Wert wird die gleiche Bit-Darstellung haben, die er bei der Kommunikationsverbindung hatte.

`raw` ist für `sensordatatype 2` nicht verfügbar.

`scale`

Datentyp: `num`

Stellen Sie den Skalierungsfaktor für Skalierdatenwerte ein. Gültige Werte sind 1, 10 oder 100. Ein Wert, der über `WriteVarArr` an ein Gerät gesendet wird, wird mit dem Skalierungsfaktor multipliziert, bevor er gesendet wird, und ein Wert, der von einem Gerät über `ReadVarArr` gelesen wird, wird durch den Skalierungsfaktor dividiert, bevor er zurückgegeben wird.

`scale` ist für `sensordatatype 2` nicht verfügbar.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.69 sensorvardata - Mehrere variable Konfigurationsdaten für die Sensorschnittstelle

Sensor Interface

Fortsetzung

`value`

Datentyp: `dnum`

Der Wert, der von dem Gerät gelesen bzw. darauf geschrieben werden soll.

Struktur

```
< data object of sensorvardata >
  < varnumber of num >
  < sensordatatype of num >
  < raw of bool >
  < scale of num >
  < value of dnum >
```

Weitere Informationen

Information über	Siehe
Mehrere Variablen von einem Gerät ablesen	*ReadVarArr - Mehrere Variablen von einem Sensorgerät ablesen auf Seite 596*
Schreiben mehrerer Variablen auf ein Gerät	*WriteVarArr - Schreiben mehrerer Variablen auf ein Sensorgerät auf Seite 1135*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration der Sensorkommunikation	*Technisches Referenzhandbuch - Systemparameter*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.70 shapedata – Weltzonen-Formdaten

Verwendung

Mit `shapedata` wird die Geometrie einer Weltzone beschrieben.

Beschreibung

Weltzonen können als 4 unterschiedliche geometrische Formen definiert werden:

- als Quader, dessen Seiten parallel zum Welt-Koordinatensystem verlaufen und der mit der Instruktion `WZBoxDef` definiert wurde,
- als Kugel, die mit der Instruktion `WZSphDef` definiert wurde,
- als Zylinder, der parallel zur z-Achse des Welt-Koordinatensystems liegt und der mit der Instruktion `WZCylDef` definiert wurde,
- als Achsenraum für Roboter und/oder externe Achsen, der mit der Instruktion `WZHomeJointDef` oder `WZLimJointDef` definiert wurde.

Die Geometrie einer Weltzone wird durch eine der vorgenannten Instruktionen definiert und die Aktion einer Weltzone wird durch die Instruktion `WZLimSup` oder `WZDOSet` definiert.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `shapedata`:

Beispiel 1:

```
VAR wzstationary pole;
VAR wzstationary conveyor;
...
PROC ...
  VAR shapedata volume;
  ...
  WZBoxDef \Inside, volume, p_corner1, p_corner2;
  WZLimSup \Stat, conveyor, volume;
  WZCylDef \Inside, volume, p_center, 200, 2500;
  WZLimSup \Stat, pole, volume;
ENDPROC
```

Ein `conveyor` wird als Quader definiert, und die Überwachung dieses Bereichs wird aktiviert. Ein `pole` wird als Zylinder definiert, und die Überwachung dieser Zone wird ebenfalls aktiviert. Wenn der Roboter einen dieser Bereiche erreicht, wird die Bewegung gestoppt.

Eigenschaften

`shapedata` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion settings*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.70 shapedata – Weltzonen-Formdaten

World Zones

Fortsetzung

Information über	Siehe
Definieren einer quaderförmigen Weltzone	*WZBoxDef - Definieren einer quaderförmigen Weltzone auf Seite 1137*
Definieren einer kugelförmigen Weltzone	*WZSphDef - Definieren einer kugelförmigen Weltzone auf Seite 1164*
Definieren einer zylinderförmigen Weltzone	*WZCylDef - Definieren einer zylinderförmigen Weltzone auf Seite 1139*
Definieren einer Weltzone für Achsen-Grundstellungen	*WZHomeJointDef - Definieren einer Weltzone für Achsen-Grundstellungen auf Seite 1153*
Definieren einer Weltzone für Grenzachsen	*WZLimJointDef - Definieren einer Weltzone mit Achsenbegrenzung auf Seite 1157*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.71 signalorigin - Gibt die Herkunft des E/A-Signals an

Verwendung

`signalorigin` stellt eine Ganzzahl mit einer symbolischen Konstanten dar.

Beschreibung

Die vordefinierten, symbolischen Konstanten des Typs `signalorigin` können zur Überprüfung der Herkunft eines E/A-Signals verwendet werden. Dies dient der Prüfung des Rückgabewerts aus der Funktion `GetSignalOrigin`.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `signalorigin`:

Beispiel 1:

```
VAR signalorigin sigorig;
VAR string signalname;
...
sigorig := GetSignalOrigin(mydo, signalname);
IF sigorig = SIGORIG_NONE THEN
  TPWrite "The signal named "+ArgName(mydo)+" can not be used";
  Stop;
ELSEIF (sigorig = SIGORIG_CFG) OR (sigorig = SIGORIG_ALIAS) THEN
  SetDO mydo, 1;
  ...
ELSE
  TPWrite "Unknown origin "+ValToStr(sigorig);
  Stop;
ENDIF
```

Die Signalherkunft wird in der Variablen `sigorig` gespeichert.

Vordefinierte Daten

Folgende Konstanten des Typs `signalorigin` sind vordefiniert:

Rückgabe-wert	Symbolische Konstante	Kommentar
0	`SIGORIG_NONE`	E/A Die E/A-Signalvariable wird in RAPID erklärt und weist keine Alias-Kupplung auf.
1	`SIGORIG_CFG`	Das Signal ist in der E/A-Konfiguration konfiguriert.
2	`SIGORIG_ALIAS`	Die E/A-Signalvariable wird in RAPID erklärt und weist eine Alias-Verkopplung zu einem in einer E/A-Konfiguration konfigurierten Signal auf.

Eigenschaften

`signalorigin` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.71 signalorigin - Gibt die Herkunft des E/A-Signals an

RobotWare - OS

Fortsetzung

Weitere Informationen

Information über	Siehe
Herkunft eines E/A-Signals	*GetSignalOrigin - Abruf von Informationen zur Herkunft eines E/A-Signals. auf Seite 1333*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.72 signalxx - Digitale und analoge Signale

Verwendung

Datentypen in `signalxx` werden für digitale und analoge Eingangs- und Ausgangssignale verwendet.

Die Namen der Signale werden in den Systemparametern definiert und müssen daher nicht im Programm definiert werden.

Beschreibung

Datentyp	Verwendet für
`signalai`	Analoge Eingangssignale
`signalao`	Analoge Ausgangssignale
`signaldi`	Digitale Eingangssignale
`signaldo`	Digitale Ausgangssignale
`signalgi`	Gruppen von digitalen Eingangssignalen
`signalgo`	Gruppen von digitalen Ausgangssignalen

Variablen des Typs `signalxo` enthalten nur einen Verweis auf das Signal. Der Wert wird über die Instruktionen `SetDO`, `SetGO` und `SetAO` gesetzt.

Variablen des Typs `signalxi` enthalten einen Verweis auf ein Signal und bieten die Möglichkeit, im entsprechenden Kontext einen Wert direkt im Programm abzurufen.

Der Wert eines Eingangssignals kann direkt im Programm gelesen werden, z. B.:

```
! Digital input
IF di1 = 1 THEN ...

! Digital group input
IF gi1 = 5 THEN ...

! Analog input
IF ai1 > 5.2 THEN ...
```

Er kann zudem in Zuordnungen verwendet werden, z. B.:

```
VAR num current_value;

! Digital input
current_value := di1;

! Digital group input
current_value := gi1;

! Analog input
current_value := ai1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.72 signalxx - Digitale und analoge Signale
RobotWare - OS
Fortsetzung

Vordefinierte Daten

Das Programm kann mithilfe der vordefinierten Signalvariablen immer auf die in den Systemparametern definierten Signale zugreifen. Beachten Sie jedoch, dass diese Signale nicht verwendbar sind, wenn andere Daten mit demselben Namen definiert wurden.

Einschränkungen

Daten vom Datentyp `signalxx` dürfen nicht im Programm definiert werden. Anderenfalls wird eine Fehlermeldung angezeigt, sobald eine Instruktion oder Funktion abgearbeitet wird, die auf dieses Signal verweist. Der Datentyp kann hingegen beim Deklarieren einer Routine als Parameter benutzt werden.

Eigenschaften

`signalxx` ist ein Datentyp mit festem Wert, der wertorientierte Operationen erlaubt.

Fehlerbehandlung

Die folgenden behebbaren Fehler werden erzeugt und können in einer Fehlerbehandlungsroutine behoben werden. Die Systemvariable `ERRNO` wird gesetzt auf:

Name	Fehlerursache
`ERR_NO_ALIASIO_DEF`	Die Signalvariable ist eine in RAPID angegebene Variable. Sie wurde nicht an ein E/A-Signal, das in der E/A-Konfiguration mit Instruktion `AliasIO` definiert wurde.
`ERR_NORUNUNIT`	Es ist kein Kontakt mit dem E/A-Gerät vorhanden.

Weitere Informationen

Information über	Siehe
Zusammenfassung von Eingangs-/Ausgangsinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Input and output signals*
Ein-/Ausgangsfunktionalität im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - I/O principles*
Konfiguration von E/A	*Technisches Referenzhandbuch - Systemparameter*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.73 socketdev - Socketgerät

Verwendung

`socketdev` (*socket device*) wird verwendet, um mit anderen Computern in einem Netzwerk oder zwischen RAPID-Tasks zu kommunizieren.

Beschreibung

Das Socket-Device ist ein Verweis auf eine Kommunikationsverbindung zu einem anderen Computer in einem Netzwerk.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `socketdev`:

Beispiel 1:

```
VAR socketdev socket1;
```

Die Variable `socket1` wird definiert und kann in einem Socket-Befehl wie beispielsweise `SocketCreate` verwendet werden.

Einschränkungen

Sie können eine beliebige Anzahl an Sockets deklarieren, es können jedoch nur 32 Sockets gleichzeitig verwendet werden.

Eigenschaften

`socketdev` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Erstellen eines neuen Sockets	*SocketCreate - Erstellen eines neuen Sockets auf Seite 754*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.74 socketstatus - Socket-Kommunikationsstatus

Verwendung

`socketstatus` wird für die Statusangabe der Socket-Kommunikation verwendet.

Beschreibung

Der Socket-Status wird mithilfe der Funktion `SocketGetStatus` abgerufen und kann für die Programmablaufsteuerung oder das Debugging verwendet werden.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `socketstatus`:

Beispiel 1:

```
VAR socketdev socket1;
VAR socketstatus state;
...
SocketCreate socket1;
state := SocketGetStatus( socket1 );
```

Der Socket-Status `SOCKET_CREATED` wird in der Variable state gespeichert.

Vordefinierte Daten

Folgende Konstanten des Typs `socketstatus` sind vordefiniert:

RAPID Konstante	Wert	Der Socket ist...
`SOCKET_CREATED`	1	Erstellt
`SOCKET_CONNECTED`	2	Mit einem Remote-Host verbundener Client
`SOCKET_BOUND`	3	An lokale Adresse und lokalen Port gebundener Server
`SOCKET_LISTENING`	4	Auf eingehende Verbindungen prüfender Server
`SOCKET_CLOSED`	5	Geschlossen

Eigenschaften

`socketstatus` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Socket-Kommunikation im Allgemeinen	*Anwendungshandbuch - Steuerungssoftware IRC5*
Abrufen des Socket-Status	*SocketGetStatus - Abrufen des aktuellen Socket-Status auf Seite 1496*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.75 speeddata - Geschwindigkeitsdaten

Verwendung

`speeddata` geben die Geschwindigkeit an, in der sich der Roboter und die externen Achsen bewegen.

Description

Geschwindigkeitsdaten definieren die Geschwindigkeit:

- in der sich der Werkzeugarbeitspunkt bewegt,
- der Umorientierung des Werkzeugs,
- in der sich externe lineare oder Rotationsachsen bewegen.

Wenn mehrere Bewegungsarten kombiniert werden, beschränkt eine der Geschwindigkeiten häufig alle Bewegungen. Die Geschwindigkeit anderer Bewegungen wird so verringert, dass die Abarbeitung aller Bewegungen gleichzeitig endet.

Die Geschwindigkeit ist auch durch die Leistung des Roboters beschränkt. Abhängig vom Robotermodell und der Bewegungsbahn gibt es dabei Unterschiede.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `speeddata`:

Beispiel 1:

```
VAR speeddata vmedium := [ 1000, 30, 200, 15 ];
```

Die Geschwindigkeitsdaten `vmedium` werden mit den folgenden Geschwindigkeiten definiert:

- `1000` mm/s für den TCP
- `30` Grad/s für die Umorientierung des Werkzeugs.
- `200` mm/s für lineare externe Achsen.
- `15` Grad/s für externe Rotationsachsen.

```
vmedium.v_tcp := 900;
```

Die Geschwindigkeit des TCP wird in `900` mm/s geändert.

Vordefinierte Daten

Einige Geschwindigkeitsdaten sind bereits im System definiert.

Vordefinierte Geschwindigkeitsdaten für die Bewegung des Roboters und der externen Achsen:

Name	TCP-Geschwindigkeit	Ausrichtung	Lineare ext. Achse	Ext. Rotationsachse
v5	5 mm/s	500°/s	5000 mm/s	1000°/s
v10	10 mm/s	500°/s	5000 mm/s	1000°/s
v20	20 mm/s	500°/s	5000 mm/s	1000°/s
v30	30 mm/s	500°/s	5000 mm/s	1000°/s
v40	40 mm/s	500°/s	5000 mm/s	1000°/s
v50	50 mm/s	500°/s	5000 mm/s	1000°/s

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.75 speeddata - Geschwindigkeitsdaten
RobotWare - OS
Fortsetzung

Name	TCP-Geschwindigkeit	Ausrichtung	Lineare ext. Achse	Ext. Rotationsachse
v60	60 mm/s	500°/s	5000 mm/s	1000°/s
v80	80 mm/s	500°/s	5000 mm/s	1000°/s
v100	100 mm/s	500°/s	5000 mm/s	1000°/s
v150	150 mm/s	500°/s	5000 mm/s	1000°/s
v200	200 mm/s	500°/s	5000 mm/s	1000°/s
v300	300 mm/s	500°/s	5000 mm/s	1000°/s
v400	400 mm/s	500°/s	5000 mm/s	1000°/s
v500	500 mm/s	500°/s	5000 mm/s	1000°/s
v600	600 mm/s	500°/s	5000 mm/s	1000°/s
v800	800 mm/s	500°/s	5000 mm/s	1000°/s
v1000	1000 mm/s	500°/s	5000 mm/s	1000°/s
v1500	1500 mm/s	500°/s	5000 mm/s	1000°/s
v2000	2000 mm/s	500°/s	5000 mm/s	1000°/s
v2500	2500 mm/s	500°/s	5000 mm/s	1000°/s
v3000	3000 mm/s	500°/s	5000 mm/s	1000°/s
v4000	4000 mm/s	500°/s	5000 mm/s	1000°/s
v5000	5000 mm/s	500°/s	5000 mm/s	1000°/s
v6000	6000 mm/s	500°/s	5000 mm/s	1000°/s
v7000	7000 mm/s	500°/s	5000 mm/s	1000°/s
vmax	i	ii	iii	iv

i Max. TCP-Geschwindigkeit für den verwendeten Robotertyp und übliche praktische TCP-Werte, angegeben durch den Systemparameter *TCP Linear Max Speed (m/s)*. Die RAPID-Funktion `MaxRobSpeed` gibt diesen Wert zurück. Wenn extrem große TCP-Werte im Werkzeugrahmen verwendet werden, können Sie Ihre eigenen `speeddata` mit größerer TCP-Geschwindigkeit als von `MaxRobSpeed` zurückgegeben erstellen und `VelSet` verwenden, um größere Geschwindigkeit zuzulassen.

ii Max. Umorientierungsgeschwindigkeit für den verwendeten Robotertyp, angegeben durch den Systemparameter *TCP Reorient Max Speed (deg/s)*. Die RAPID-Funktion `MaxRobReorientSpeed` gibt diesen Wert zurück.

iii Max. Lineargeschwindigkeit für zusätzliche Achsen, angegeben durch den Systemparameter *Ext. Axis Linear Max Speed (m/s)*. Die RAPID-Funktion `MaxExtLinearSpeed` gibt diesen Wert zurück.

iv Max. Rotationsgeschwindigkeit für zusätzliche Achsen, angegeben durch den Systemparameter *Ext. Axis Rotational Max Speed (deg/s)*. Die RAPID-Funktion `MaxExtReorientSpeed` gibt diesen Wert zurück.

Vordefinierte "speeddata" für sich bewegende externe Rotationsachsen mit der Instruktion `MoveExtJ`.

Name	TCP-Geschwindigkeit	Ausrichtung	Lineare ext. Achse	Ext. Rotationsachse
vrot1	0 mm/s	0°/s	0 mm/s	1°/s
vrot2	0 mm/s	0°/s	0 mm/s	2°/s
vrot5	0 mm/s	0°/s	0 mm/s	5°/s
vrot10	0 mm/s	0°/s	0 mm/s	10°/s
vrot20	0 mm/s	0°/s	0 mm/s	20°/s
vrot50	0 mm/s	0°/s	0 mm/s	50°/s

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Name	TCP-Geschwindigkeit	Ausrichtung	Lineare ext. Achse	Ext. Rotationsachse
vrot100	0 mm/s	0°/s	0 mm/s	100°/s

Vordefinierte "speeddata" für sich bewegende lineare externe Achsen mit der Instruktion `MoveExtJ`.

Name	TCP-Geschwindigkeit	Ausrichtung	Lineare ext. Achse	Ext. Rotationsachse
vlin10	0 mm/s	0°/s	10 mm/s	0°/s
vlin20	0 mm/s	0°/s	20 mm/s	0°/s
vlin50	0 mm/s	0°/s	50 mm/s	0°/s
vlin100	0 mm/s	0°/s	100 mm/s	0°/s
vlin200	0 mm/s	0°/s	200 mm/s	0°/s
vlin500	0 mm/s	0°/s	500 mm/s	0°/s
vlin1000	0 mm/s	0°/s	1000 mm/s	0°/s

Komponenten

`v_tcp`

velocity tcp

Datentyp: `num`

Die Geschwindigkeit des TCP (Werkzeugarbeitspunkts) in mm/s.

Wenn ein stationäres Werkzeug oder koordinierte externe Achsen verwendet werden, wird die Geschwindigkeit relativ zum Werkobjekt angegeben.

`v_ori`

velocity orientation

Datentyp: `num`

Die Geschwindigkeit der Umorientierung des TCP in Grad/s.

Wenn ein stationäres Werkzeug oder koordinierte externe Achsen verwendet werden, wird die Geschwindigkeit relativ zum Werkobjekt angegeben.

`v_leax`

velocity linear external axes

Datentyp: `num`

Die Geschwindigkeit von linearen externen Achsen in mm/s.

`v_reax`

velocity rotational external axes

Datentyp: `num`

Die Geschwindigkeit von externen Rotationsachsen in Grad/s.

Struktur

```
< dataobject of speeddata >
  < v_tcp of num >
  < v_ori of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.75 speeddata - Geschwindigkeitsdaten
RobotWare - OS
Fortsetzung

```
< v_leax of num >
< v_reax of num >
```

Einschränkungen

Bei sehr geringer Geschwindigkeit sollte jede Bewegung so kurz sein, dass für eine Interpolation weniger als 240 Sekunden verfügbar sind.

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Motion*
Bewegung/Geschwindigkeit im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - Positioning during program execution*
Definieren der Höchstgeschwindigkeit	*VelSet - Ändern der programmierten Geschwindigkeit auf Seite 1038*
TCP-Höchstgeschwindigkeit für diesen Roboter	*MaxRobSpeed - Höchstgeschwindigkeit des Roboters auf Seite 1394*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.76 stoppointdata - Stopppunktdaten

Verwendung

`stoppointdata` dient der Angabe, wie eine Position erreicht wird, d. h., wie nahe sich die Achsen an der programmierten Position befinden müssen, bevor sie sich zur nächsten Position bewegen.

Beschreibung

Eine Position kann als Stopppunkt oder Fly-By-Punkt festgelegt werden.

Ein Fly-By-Punkt bedeutet, dass die programmierte Position nie erreicht wird. In der Bewegungsinstruktion wird eine Zone angegeben, um eine Zonenbahn zu definieren. Die Bewegungsrichtung wird vor Erreichen der programmierten Position durch die Zonenbahn zur nächsten Position gelenkt (siehe Datentyp `zonedata`).

Ein Stopppunkt bedeutet, dass der Roboter und externe Achsen die angegebene Position erreichen müssen, bevor der Roboter/die externen Achsen mit der nächsten Bewegung fortfahren. Der Stopppunkt gilt als erreicht, sobald die Konvergenzkriterien des Punkts erfüllt sind. Die Konvergenzkriterien sind Geschwindigkeit und Position. Es ist auch möglich, Zeitkriterien festzulegen. Weitere Informationen über den Stopppunkt `fine` finden Sie auch unter dem Datentyp `zonedata`.

Durch `stoppointdata` können drei Typen von Stopppunkten definiert werden:

- Der Stopppunkt-Typ **in position** wird als Prozentsatz der Konvergenzkriterien (Position und Geschwindigkeit) für den vordefinierten Stopppunkt `fine` festgelegt. Dieser Typ verwendet auch eine Mindest- und Höchstdauer. Der Roboter wartet mindestens die Mindestdauer und maximal die Höchstdauer darauf, dass die Positions- und Geschwindigkeitskriterien erfüllt sind.
- Beim Stopppunkt-Typ **stop time** wartet der Roboter stets die vorgegebene Dauer am Stopppunkt.
- Beim Stopppunkt-Typ **follow time** handelt es sich um einen besonderen Typ eines Stopppunkts, der der Koordination der Roboterbewegung mit einem Conveyor dient.

`stoppointdata` bestimmt auch, wie die Bewegung mit der RAPID-Abarbeitung synchronisiert wird. Wenn die Bewegung synchronisiert ist, wartet die RAPID-Abarbeitung auf ein „in pos“-Ereignis, sobald sich der Roboter in Position befindet. Wenn die Bewegung nicht synchronisiert ist, erhält die RAPID-Abarbeitung ein „prefetch“-Ereignis beinahe eine halbe Sekunde, bevor der physische Roboter die programmierte Position erreicht. Wenn die Programmabarbeitung ein „in pos“- oder „prefetch“-Ereignis erhält, fährt sie mit der nächsten Instruktion fort. Wenn das „prefetch“-Ereignis eintritt, muss der Roboter noch eine längere Strecke zurücklegen. Wenn das „in pos“-Ereignis eintritt, befindet sich der Roboter nahe an der programmierten Position.

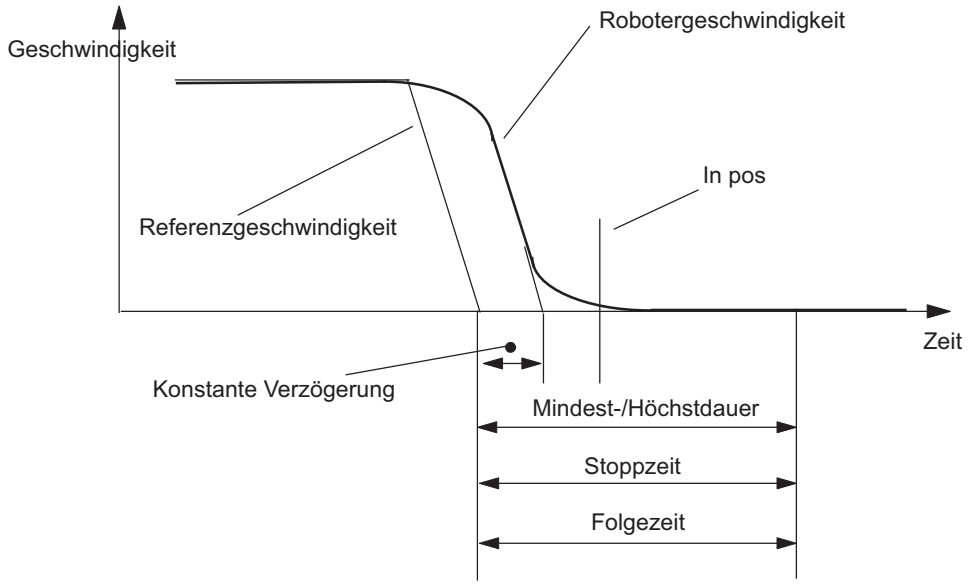
Für die Typen **stop time** und **follow time** beginnt die Abarbeitung der nächsten Instruktion zum gleichen Zeitpunkt wie das Herunterzählen von "stop time" bzw. "follow time". Für den Typ **in position** wird jedoch die nächste Instruktion abgearbeitet, wenn die Konvergenzkriterien erfüllt sind.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.76 stoppointdata - Stopppunktdaten
RobotWare - OS
Fortsetzung

Wenn Sie Bewegungsinstruktionen mit dem Argument `\Conc` verwenden, erfolgt überhaupt keine Synchronisierung, daher wird die aktuelle Bewegungsinstruktion sofort abgearbeitet.



xx0500002374

Obige Abbildung illustriert die Endphase einer Bewegung zu einem Stopppunkt. Die Robotergeschwindigkeit nimmt nicht linear ab. Der Roboterservo ist dem physischen Roboter stets voraus. Dies wird in obiger Abbildung als konstante Verzögerung dargestellt. Die konstante Verzögerung beträgt etwa 0,1 Sekunden. Für die Zeitelemente von `stoppointdata` dient die Referenzgeschwindigkeit als Trigger. Wenn die Referenzgeschwindigkeit null beträgt, beginnt die Messung der Zeit. Daher umfasst die Dauer in den Zeitelementen stets die konstante Verzögerung. Die Verwendung von Werten, die kleiner als die konstante Verzögerung sind, ist deshalb sinnlos.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `stoppointdata`:

Inpos

```
VAR stoppointdata my_inpos := [ inpos, TRUE, [ 25, 40, 0.1, 5], 0,
     0, "", 0, 0];
MoveL *, v1000, fine \Inpos:=my_inpos, grip4;
```

Die Stopppunktdaten `my_inpos` werden mit den folgenden Eigenschaften definiert:

- Der Typ des Stopppunkts ist "in Position", `inpos`.
- Der Stopppunkt wird mit der RAPID-Programmabarbeitung synchronisiert, `TRUE`.
- Das Abstandskriterium des Stopppunkts beträgt 25 % des Abstands, der für den Stopppunkt `fine` definiert ist, nämlich 25.
- Das Geschwindigkeitskriterium des Stopppunkts beträgt 40 % des Abstands, der für den Stopppunkt `fine` definiert ist, nämlich 40.
- Die minimale Wartezeit für die Konvergenz beträgt 0,1 s, 0,1.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Die maximale Wartezeit für die Konvergenz beträgt 5 s, 5.

Der Roboter bewegt sich zur programmierten Position, bis das Positions- oder Geschwindigkeitskriterium erfüllt ist.

```
my_inpos.inpos.position := 40;
MoveL *, v1000, fine \Inpos:=my_inpos, grip4;
```

Das Abstandskriterium des Stopppunkts wird auf 40 % angepasst.

Stoppzeit

```
VAR stoppointdata my_stoptime := [ stoptime, FALSE, [ 0, 0, 0, 0],
     1.45, 0, "", 0, 0];
MoveL *, v1000, fine \Inpos:=my_stoptime, grip4;
```

Die Stopppunktdaten `my_stoptime` werden mit den folgenden Eigenschaften definiert:

- Der Typ des Stopppunkts ist "Stoppzeit", `stoptime`.
- Der Stopppunkt wird nicht mit der RAPID-Programmabarbeitung synchronisiert, `FALSE`.
- Die Wartezeit an der Position beträgt 1,45 s.

Der Roboter bewegt sich zur programmierten Position, bis das prefetch-Ereignis eintritt. Die nächste RAPID-Instruktion wird abgearbeitet. Wenn es sich um eine Bewegungsinstruktion handelt, stoppt der Roboter 1,45 Sekunden lang, bevor die nächste Bewegung beginnt.

```
my_stoptime.stoptime := 6.66;
MoveL *, v1000, fine \Inpos:=my_stoptime, grip4;
```

Die Stoppzeit des Stopppunkts wird auf 6,66 s eingestellt. Wenn es sich bei der nächsten RAPID-Instruktion um eine Bewegungsinstruktion handelt, stoppt der Roboter für 6,66 s.

Folgezeit

```
VAR stoppointdata my_followtime := [ fllwtime, TRUE, [ 0, 0, 0,
     0], 0, 0.5, "", 0, 0];
MoveL *, v1000, z10 \Inpos:=my_followtime, grip6\wobj:=conveyor1;
```

Die Stopppunktdaten `my_followtime` werden mit den folgenden Eigenschaften definiert:

- Der Typ des Stopppunkts ist "Folgezeit", `fllwtime`.
- Der Stopppunkt wird mit der RAPID-Programmabarbeitung synchronisiert, `TRUE`.
- Die Folgezeit des Stopppunkts beträgt 0,5 s, 0,5.

Der Roboter folgt dem Conveyor für die Dauer von 0,5 s, bevor er ihn mit der Zone 10 mm, `z10`, verlässt.

```
my_followtime.followtime := 0.4;
```

Die Folgezeit des Stopppunkts wird auf 0,4 s eingestellt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.76 stoppointdata - Stopppunktdaten
RobotWare - OS
Fortsetzung

Vordefinierte Daten

Einige Bremspunktdaten sind bereits im System definiert.

Stopppunkte des Typs in position

Name	Progsynch	Stelle	Geschwindigkeit	Min. Zeit	Max. Zeit	Stoppzeit	Folgezeit
inpos20	TRUE	20%	20%	0 s	2 s	-	-
inpos50	TRUE	50%	50%	0 s	2 s	-	-
inpos100	TRUE	100%	100%	0 s	2 s	-	-

(`inpos100` verfügt über dieselben Konvergenzkriterien wie Stopppunkt `fine`)

Stopppunkte des Typs stop time

Name	Progsynch	Stelle	Geschwindigkeit	Min. Zeit	Max. Zeit	Stoppzeit	Folgezeit
stoptime0_5	FALSE	-	-	-	-	0.5 s	-
stoptime1_0	FALSE	-	-	-	-	1.0 s	-
stoptime1_5	FALSE	-	-	-	-	1.5 s	-

Stopppunkte des Typs follow time

Name	Progsynch	Stelle	Geschwindigkeit	Min. Zeit	Max. Zeit	Stoppzeit	Folgezeit
fllwtime0_5	TRUE	-	-	-	-	-	0.5 s
fllwtime1_0	TRUE	-	-	-	-	-	1.0 s
fllwtime1_5	TRUE	-	-	-	-	-	1.5 s

Komponenten

`type`

type of stop point

Datentyp: `stoppoint`

Die folgende Tabelle definiert den Typ des Stopppunkts (`stoppoint`).

`1 (inpos)`	Die Bewegung endet mit einem Stopppunkt des Typs inposition. Aktiviert das Element `inpos` in `stoppointdata`. Die Zonendaten in der Instruktion werden nicht verwendet; verwenden Sie `fine` oder `z0`.
`2 (stoptime)`	Die Bewegung endet mit einem Stopppunkt des Typs stoptime. Aktiviert das Element `stoptime` in `stoppointdata`. Die Zonendaten in der Instruktion werden nicht verwendet; verwenden Sie `fine` oder `z0`.
`3 (followtime)`	Die Bewegung endet mit einem Feinpunkt des Typs followtime. Die Zonendaten in der Instruktion werden verwendet, wenn der Roboter den Conveyor verlässt. Aktiviert das Element `followtime` in `stoppointdata`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp `stoppoint` ist ein Alias-Datentyp für `num`. Mit ihm wird der Typ des Stopppunkts gewählt und welche Datenelemente in `stoppointdata` benutzt werden sollen. Vordefinierte Konstanten sind:

Wert	Symbolische Konstante	Kommentar
1	`inpos`	"In Position"-Typ
2	`stoptime`	"Stoppzeit"-Typ
3	`fllwtime`	"Folgezeit"-Typ

`progsynch`

program synchronization

Datentyp: `bool`

Synchronisierung mit RAPID-Programmabarbeitung

- `TRUE`: Die Bewegung wird mit der RAPID-Abarbeitung synchronisiert. Das Programm arbeitet die nächste Instruktion erst ab, wenn der Stopppunkt erreicht wurde.
- `FALSE`: Die Bewegung wird nicht mit der RAPID-Abarbeitung synchronisiert. Das Programm beginnt die Abarbeitung der nächsten Instruktion, bevor der Stopppunkt erreicht wurde.

Wenn Sie Bewegungsinstruktionen mit dem Argument `\Conc` verwenden, erfolgt unabhängig von den Daten in `progsynch` keinerlei Synchronisierung, daher wird die aktuelle Bewegungsinstruktion immer sofort abgearbeitet.

`inpos.position`

position condition for TCP

Datentyp: `num`

Die Positionsbedingung (der Radius) für den TCP als Prozentsatz eines normalen `fine`-Stopppunkts.

`inpos.speed`

speed condition for TCP

Datentyp: `num`

Die Geschwindigkeitsbedingung für den TCP als Prozentsatz eines normalen `fine`-Stopppunkts.

`inpos.mintime`

minimum wait time

Datentyp: `num`

Die minimale Wartezeit in Sekunden vor "in position". Der Roboter wartet dabei an dem Punkt mindestens für die angegebene Zeitdauer. Der Höchstwert beträgt 20,0 Sekunden.

`inpos.maxtime`

maximum wait time

Datentyp: `num`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.76 stoppointdata - Stopppunktdaten
RobotWare - OS
Fortsetzung

Die maximale Wartezeit in Sekunden, bis die Konvergenzkriterien erfüllt sind. Damit wird gewährleistet, dass der Roboter nicht an dem Punkt hängen bleibt, wenn Geschwindigkeits- und Positionsbedingungen zu knapp angesetzt sind. Der Höchstwert beträgt 20,0 Sekunden.

`stoptime`

stop time

Datentyp: `num`

Die Stillstanddauer in Sekunden, bevor der TCP von dem Punkt aus die nächste Bewegung beginnt. Gültiger Bereich 0-20 s, Auflösung 0,001 s.

`followtime`

follow time

Datentyp: `num`

Die Dauer in Sekunden, nach der der TCP dem Conveyor folgt. Gültiger Bereich 0-20 s, Auflösung 0,001 s.

`signal`

Datentyp: `string`

Reserviert für zukünftige Verwendung.

`relation`

Datentyp: `opnum`

Reserviert für zukünftige Verwendung.

`checkvalue`

Datentyp: `num`

Reserviert für zukünftige Verwendung.

Struktur

```
< data object of stoppointdata >
  < type of stoppoint >
  < progsynch of bool >
  < inpos of inposdata >
    < position of num >
    < speed of num >
    < mintime of num >
    < maxtime of num >
  < stoptime of num >
  < followtime of num >
  < signal of string >
  < relation of opnum >
  < checkvalue of num >
```

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Information über	Siehe
Bewegungen/Bahnen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - Positioning during program execution*
Stopp- oder Fly-By-Punkte	*zonedata - Zonendaten auf Seite 1860*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.77 string – Zeichenfolgen

Verwendung

`string` wird für Zeichenfolgen verwendet.

Beschreibung

Eine Zeichenfolge (string) besteht aus einer Reihe von Zeichen (maximal 80), die von doppelten Hochkommas ("") umgeben sind, z. B. "Das ist eine Zeichenfolge".

Wenn die Hochkommas in die Zeichenfolge aufgenommen werden sollen, müssen sie zweimal eingegeben werden, z. B. "Diese Zeichenfolge enthält ein ""-Zeichen".

Wenn umgekehrte Schrägstriche in die Zeichenfolge aufgenommen werden sollen, müssen sie zweimal eingegeben werden, z. B. „Diese Zeichenfolge enthält ein \\-Zeichen“.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `string`:

Beispiel 1:

```
VAR string text;
...
text := "start welding pipe 1";
TPWrite text;
```

Der Text `start welding pipe 1` wird am FlexPendant ausgegeben.

Vordefinierte Daten

Eine Reihe vordefinierter string-Konstanten ist im System verfügbar und kann mit string-Funktionen verwendet werden. Siehe das Beispiel `StrMemb`.

Name	Zeichensatz
STR_DIGIT	<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
STR_UPPER	<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï | 1)| Ñ | Ò | Ó | Ô | Õ | Ö | Ø | Ù | Ú | Û | Ü | 2) | 3)
STR_LOWER	<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï | 1) | ñ | ò | ó | ô | õ | ö | ø | ù | ú | û | ü | 2) | 3) | ß | ÿ-
STR_WHITE	<blank character> ::=

1) Isländischer Buchstabe eth.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

2) Buchstabe Y mit Akzent.

3) Isländischer Buchstabe thorn.

Die folgenden Konstanten sind bereits im System definiert:

```
CONST string diskhome := "HOME:";

! For old programs from S4C system
CONST string ram1disk := "HOME:";

CONST string disktemp := "TEMP:";

CONST string flp1 := "flp1:";

CONST string stSpace := " ";

CONST string stEmpty := "";
```

Einschränkungen

Eine Zeichenfolge kann aus 0 bis 80 Zeichen bestehen, ggf. einschließlich zusätzlicher Hochkommas oder umgekehrter Schrägstriche.

Eine Zeichenfolge kann beliebige durch ISO 8859-1 (Latin-1) festgelegte Zeichen sowie Steuerzeichen (Nicht-ISO8859-1 (Latin-1) im numerischen Codebereich 0-255) enthalten.

Weitere Informationen

Information über	Siehe
Operationen mit Zeichenfolgen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Expressions*
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Basic elements*
Instruktion mit Zeichensatz	*StrMemb - Prüfen, ob ein Zeichen einer Zeichenmenge angehört auf Seite 1529*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.78 stringdig - Zeichenfolge, die nur Ziffern enthält

Verwendung

`stringdig` dient zum Darstellen großer positiver Ganzzahlen in einer Zeichenfolge, die nur Ziffern enthält.

Dieser Datentyp wurde eingeführt, weil der Datentyp `num` positive Ganzzahlen größer als 8.388.608 nicht genau darstellen kann.

Beschreibung

`stringdig` darf nur aus einer Anzahl von Ziffern von 0 bis 9, innerhalb von Anführungszeichen (" ") bestehen, z. B. "0123456789".

Der Datentyp `stringdig` kann positive Ganzzahlen bis 4.294.967.295 behandeln.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `stringdig`:

Beispiel 1:

```
VAR stringdig digits1;
VAR stringdig digits2;
VAR bool flag1;
...
digits1 ="09000000";
digits2 = "9000001";
flag1 := StrDigCmp (digits1, LT, digits2);
```

The data `flag1` wird auf `TRUE` gesetzt, weil `09000000` kleiner als `9000001` ist.

Eigenschaften

`stringdig` ist ein Alias-Datentyp für `string` und übernimmt daher den Großteil seiner Eigenschaften.

Weitere Informationen

Information über	Siehe
Zeichenfolgenwerte	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Basic elements*
Zeichenfolgen	*string – Zeichenfolgen auf Seite 1806*
Numerische Werte	*num - Numerische Werte auf Seite 1740*
Vergleichsoperator	*opnum - Vergleichsoperator auf Seite 1743StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten auf Seite 1518*
Vergleichen von Zeichenfolgen, die nur Ziffern enthalten	*StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten auf Seite 1518*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.79 supervtimeouts - Zeitüberschreitungen der Handshake-Überwachung

Verwendung

`supervtimeouts` wird verwendet, um Zeitüberschreitungen für die Handshake-Überwachung des CAP zu definieren.

`supervtimeouts` ist eine Komponente von `capdata` und definiert die Zeitüberschreitungen für die folgenden Phasen der Handshake-Überwachung des CAP:

- PRE
- END_PRE und START
- END MAIN und START_POST1
- END_POST1 und START_POST2
- END_POST2

Wenn der Parameter auf 0 eingestellt ist, gibt es keine Zeitüberschreitung.

Komponenten

pre_cond

Datentyp: `num`

Zeitüberschreitung (in Sekunden), in der die Bedingungen der PRE-Phasen erfüllt sein müssen.

start_cond

Datentyp: `num`

Zeitüberschreitung (in Sekunden) in der die Bedingungen der Phasen END_PRE und START erfüllt sein müssen.

end_main_cond

Datentyp: `num`

Zeitüberschreitung (in Sekunden) in der die Bedingungen der Phasen END_MAIN und START_POST1 erfüllt sein müssen.

end_post1_cond

Datentyp: `num`

Zeitüberschreitung (in Sekunden) in der die Bedingungen der Phasen END_POST1 und START_POST2 erfüllt sein müssen.

end_post2_cond

Datentyp: `num`

Zeitüberschreitung (in Sekunden), in der die Bedingungen der END_POST2-Phasen erfüllt sein müssen.

Syntax

```
< data object of supervtimeouts >
  < pre_cond of num >
  < start_cond of num >
  < end_main_cond of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.79 supervtimeouts - Zeitüberschreitungen der Handshake-Überwachung

Continuous Application Platform (CAP)

Fortsetzung

```
< end_post1_cond of num >
< end_post2_cond of num >
```

Weitere Informationen

	Beschrieben in:
`capdata` Datentyp	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.80 switch – Optionale Parameter

Verwendung

`switch` wird für optionale Parameter verwendet.

Beschreibung

Der Spezialtyp `switch` darf (nur) optionalen Parametern zugewiesen werden. Er bietet die Möglichkeit, Argumente zu wechseln, die nur durch Namen (nicht Werte) angegeben sind. An einen switch-Parameter kann kein Wert übertragen werden. Ein switch-Parameter kann nur verwendet werden, wenn sein Vorhandensein über die vordefinierte Funktion `Present` geprüft wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `switch`:

Beispiel 1:

```
PROC my_routine(\switch on | switch off)
....
IF Present (off) THEN
....
ENDIF
ENDPROC
```

Je nach den Argumenten, die vom Aufrufer von `my_routine` verwendet werden, lässt sich der Programmablauf steuern.

Eigenschaften

`switch` ist ein Datentyp ohne einen Wert (Non-Value) und kann nicht in wertorientierten Operationen verwendet werden.

Weitere Informationen

Information über	Siehe
Parameter	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Routines*.
Überprüfen, ob ein optionaler Parameter vorhanden ist	*Present - Prüft, ob ein optionaler Parameter verwendet wird auf Seite 1448*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.81 symnum - Symbolische Zahl

Verwendung

symnum (*Symbolic Number*) stellt eine Ganzzahl mit einer symbolischen Konstanten dar.

Beschreibung

Eine symnum-Konstante dient der Prüfung des Rückgabewerts aus der Funktion OpMode und RunMode.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp symnum:

Beispiel 1:

```
IF RunMode() = RUN_CONT_CYCLE THEN
..
ELSE
..
ENDIF
```

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps symnum sind vordefiniert und können beim Prüfen der zurückgegebenen Werte von den Funktionen OpMode und RunMode verwendet werden.

Wert	Symbolische Konstante	Kommentar
0	RUN_UNDEF	Undefinierter Abarbeitungsmodus
1	RUN_CONT_CYCLE	Kontinuierlicher oder zyklischer Abarbeitungsmodus
2	RUN_INSTR_FWD	Vorwärtsabarbeitungsmodus für Instruktion
3	RUN_INSTR_BWD	Rückwärtsabarbeitungsmodus für Instruktion
4	RUN_SIM	Simulierter Abarbeitungsmodus
5	RUN_STEP_MOVE	Bewegungsinstruktionen in Vorwärtsabarbeitung und logische Instruktionen in kontinuierlichem Abarbeitungsmodus

Wert	Symbolische Konstante	Kommentar
0	OP_UNDEF	Undefinierter Betriebsmodus
1	OP_AUTO	Automatischer Betriebsmodus
2	OP_MAN_PROG	Einrichtbetrieb max. 250 mm/s
3	OP_MAN_TEST	Einrichtbetrieb, volle Geschwindigkeit 100 %

Eigenschaften

Symnum ist ein Alias-Datentyp für num und übernimmt daher dessen Eigenschaften.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Weitere Informationen

Information über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.82 syncident - Identität für Synchronisierungspunkte

Verwendung

`syncident` (*synchronization identity*) wird für die Angabe des Namens eines Synchronisierungspunkts verwendet. Der Name eines Synchronisierungspunkts ist der Name (die Identität) des deklarierten Datentyps `syncident`.

Beschreibung

`syncident` dient zur Identifikation eines Punktes im Programm, an dem die aktuelle Programmtask darauf wartet, dass die koordinierten Programmtasks denselben Synchronisierungspunkt erreichen.

Der Datenname (die Identität) des Typs `syncident` muss in allen koordinierten Programmtasks identisch sein.

Der Datentyp syncident wird in den Instruktionen `WaitSyncTask`, `SyncMoveOn` und `SyncMoveOff` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `syncident`:

Beispiel 1:

Programmbeispiel in der Programmtask `ROB1`

```
PERS tasks task_list{3} := [ ["STN1"], ["ROB1"], ["ROB2"] ];
VAR syncident sync1;

WaitSyncTask sync1, task_list;
```

Bei der Abarbeitung der Instruktion `WaitSyncTask` in der Programmtask `ROB1` wird die Abarbeitung in dieser Programmtask angehalten, bis die anderen Programmtasks `STN1` und `ROB2` die entsprechende `WaitSyncTask` mit demselben Synchronisierungspunkt (Treffpunkt) `sync1` erreicht haben.

Struktur

`syncident` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Angeben von koordinierten Programmtasks	*tasks – RAPID-Programmtasks auf Seite 1819*
Warten auf Synchronisierungspunkt mit anderen Tasks	*WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten auf Seite 1089*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.83 System data – Aktuelle Einstellungen der RAPID-Systemdaten

Verwendung

`System data` zeigt die aktuellen Einstellungen der RAPID-Systemdaten an, wie z. B. aktuelle Bewegungseinstellungen, die aktuelle Fehlernummer `ERRNO`, die aktuelle Interrupt-Nummer `INTNO` usw.

Das Programm kann auf diese Daten zugreifen und sie lesen. An ihnen kann der aktuelle Status abgelesen werden, z. B. die aktuelle Programmverschiebung.

`C_MOTSET`

Die Variable `C_MOTSET` des Datentyps `motsetdata` gibt die aktuellen Bewegungseinstellungen an:

Beschreibung	Datentyp	Geändert von	Siehe auch:
Aktuelle Bewegungseinstellungen, d. h.:	`motsetdata`	Instruktionen	*motsetdata - Daten für Bewegungseinstellungen auf Seite 1733*
Geschwindigkeitsoverride und Höchstgeschwindigkeit		`VelSet`	*VelSet - Ändern der programmierten Geschwindigkeit auf Seite 1038*
Geschwindigkeitsoverride		`AccSet`	*AccSet - Verringern der Beschleunigung auf Seite 25*
Bewegungen in der Nähe von singulären Punkten		`SingArea`	*SingArea - Definieren der Interpolation in der Nähe singulärer Punkte auf Seite 736*
Lineare Konfigurationssteuerung Achsenkonfigurationssteuerung		`ConfL` `ConfJ`	*ConfL - Überwachen der Konfiguration während linearer Bewegung auf Seite 158* *ConfJ - Steuern der Konfiguration bei achsenweiser Bewegung auf Seite 156*
Bahnauflösung		`PathResol`	*PathResol - Override der Bahnauflösung auf Seite 534*
Einstellen der Bewegungsüberwachung		`MotionSup`	*MotionSup - Deaktivieren/Aktivieren der Bewegungsüberwachung auf Seite 379*
Reduktion der TCP-Beschleunigung/Verzögerung entlang der Bewegungsbahn		`PathAccLim`	*PathAccLim - Beschränken der TCP-Beschleunigung entlang der Bahn auf Seite 507*
Ändern der Werkzeugorientierung bei der kreisförmigen Interpolation		`CirPathMode`	*CirPathMode - Werkzeugumorientierung auf kreisförmiger Bahn auf Seite 133*
Reduktion der Nutzlast-Beschleunigung in Welt-Koordinatensystem		`WorldAccLim`	*WorldAccLim - Steuern der Beschleunigung im Welt-Koordinatensystem auf Seite 1112*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.83 System data – Aktuelle Einstellungen der RAPID-Systemdaten

RobotWare - OS

Fortsetzung

C_PROGDISP

Die Variable C_PROGDISP des Datentyps progdisp gibt die aktuelle Programmverschiebung und den Offset der externen Achsen an:

Beschreibung	Datentyp	Geändert von	Siehe auch:
Aktuelle Programmverschiebung für Roboterachsen	progdisp	Instruktionen:	*progdisp - Programmverschiebung auf Seite 1761*
		PDispSet	*PDispSet - Aktivieren einer Programmverschiebung in einem bekannten Koordinatensystem auf Seite 542*
		PDispOn	*PDispOn - Aktivieren der Programmverschiebung auf Seite 537*
		PDispOff	*PDispOff - Deaktivieren der Programmverschiebung auf Seite 536*
Aktueller Offset der externen Achsen		EOffsSet	*EOffsSet - Aktivieren eines Offsets für zusätzliche Achsen durch bekannte Werte auf Seite 198*
		EOffsOn	*EOffsOff - Aktivieren eines Offsets für zusätzliche Achsen auf Seite 195*
		EOffsOff	*EOffsOff - Deaktivieren eines Offsets für zusätzliche Achsen auf Seite 194*

ERRNO

Die Variable ERRNO des Datentyps errnum gibt die aktuelle Fehlernummer an:

Beschreibung	Datentyp	Geändert von	Siehe auch:
Der zuletzt aufgetretene Fehler	errnum	Das System	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Error recovery* *intnum - Interrupt-Identität auf Seite 1716*

INTNO

Die Variable INTNO des Datentyps intnum gibt die aktuelle Interrupt-Nummer an:

Beschreibung	Datentyp	Geändert von	Siehe auch:
Der zuletzt aufgetretene Interrupt	intnum	Das System	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Interrupts* *intnum - Interrupt-Identität auf Seite 1716*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ROB_ID

Die Variable ROB_ID des Datentyps mecunit enthält einen Verweis auf den TCP-Roboter (falls vorhanden) in der aktuellen Programmtask.

Beschreibung	Datentyp	Geändert von	Siehe auch:
Verweis auf den Roboter (falls vorhanden) in der aktuellen Programmtask. Immer prüfen vor der Verwendung mit TaskRunRob()	mecunit	Das System	*mecunit – Mechanische Einheit auf Seite 1731*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.84 taskid – Taskidentifizierung

Verwendung

`taskid` wird zur Identifizierung verfügbarer Programmtasks im System verwendet.

Die Namen der Programmtasks werden in den Systemparametern definiert und müssen daher nicht im Programm definiert werden.

Beschreibung

Daten des Typs `taskid` enthalten nur einen Verweis auf die Programmtask.

Vordefinierte Daten

Das Programm kann immer auf die in den Systemparametern definierten Programmtasks (installierte Daten) zugreifen.

Für alle Programmtasks im System stehen vordefinierte Variablen des Datentyps `taskid` zur Verfügung. Die Variablenidentität lautet „taskname“+„Id“ (die Variablenidentität für die Task `T_ROB1` lautet z. B. `T_ROB1Id`, `T_ROB2` - `T_ROB2Id` usw.).

Einschränkungen

Daten des Typs `taskid` dürfen nicht im Programm definiert werden. Der Datentyp kann hingegen beim Deklarieren einer Routine als Parameter benutzt werden.

Eigenschaften

`taskid` ist ein Datentyp ohne einen Wert (Non-Value). Das heißt, Daten dieses Typs erlauben keine wertorientierten Operationen.

Weitere Informationen

Information über	Siehe
Speichern von Programmmodulen	*Save - Speichern eines Programmmoduls auf Seite 653*
Konfiguration von Programmtasks	*Technisches Referenzhandbuch - Systemparameter*
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.85 tasks – RAPID-Programmtasks

Verwendung

`tasks` wird zur Angabe mehrerer RAPID-Programmtasks verwendet.

Beschreibung

Für die Angabe mehrerer RAPID-Programmtasks kann der Name jeder Task als Zeichenfolge wiedergegeben werden. Ein Datenfeld des Datentyps `tasks` kann dann sämtliche Tasknamen enthalten.

Diese Taskliste kann dann in den Instruktionen `WaitSyncTask` und `SyncMoveOn` verwendet werden.

Hinweis

Die Daten für die obigen Instruktionen müssen als systemweite `PERS`-Variablen definiert sein, die in allen koordinierten Tasks verfügbar sind.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `tasks`:

Beispiel 1:

Programmbeispiel in der Programmtask `T_ROB1`

```
PERS tasks task_list{3} := [ ["T_STN1"], ["T_ROB1"], ["T_ROB2"] ];
VAR syncident sync1;

WaitSyncTask sync1, task_list;
```

Bei der Abarbeitung der Instruktion `WaitSyncTask` in der Programmtask `T_ROB1` wird die Abarbeitung in dieser Programmtask angehalten, bis die anderen Programmtasks `T_STN1` und `T_ROB2` die entsprechende `WaitSyncTask` mit demselben Synchronisierungspunkt (Treffpunkt) `sync1` erreicht haben.

Komponenten

Der Datentyp hat die folgenden Komponenten.

`taskname`

Datentyp: `string`

Der Name einer RAPID-Programmtask, in einer Zeichenfolge angegeben

Struktur

```
<dataobject of tasks>
  <taskname of string>
```

Weitere Informationen

Information über	Siehe
Identität für Synchronisierungspunkte	*syncident - Identität für Synchronisierungspunkte auf Seite 1814*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.85 tasks – RAPID-Programmtasks

Multitasking

Fortsetzung

Information über	Siehe
Warten auf Synchronisierungspunkt mit anderen Tasks	*WaitSyncTask – Am Synchronisierungspunkt auf andere Programmtasks warten auf Seite 1089*
Beginn von koordinierten Synchronbewegungen	*SyncMoveOn - Beginn von koordinierten Synchronbewegungen auf Seite 863*
Ende von koordinierten Synchronbewegungen	*SyncMoveOff - Beenden von koordinierten Synchronbewegungen auf Seite 857*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.86 testsignal – Prüfsignal

Verwendung

Der Datentyp `testsignal` wird beim Test des Roboterbewegungssystems verwendet.

Beschreibung

Eine Reihe vordefinierter Prüfsignale steht im Robotersystem zur Verfügung. Der Datentyp `testsignal` vereinfacht die Programmierung der Instruktion `TestSignDefine`.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `testsignal`:

Beispiel 1:

```
TestSignDefine 2, speed, Orbit, 2, 0;
```

Die vordefinierte Konstante `speed` wird verwendet, um die aktuelle Geschwindigkeit von Achse `2` am Manipulator `orbit` abzulesen.

Beispiel 2:

```
TestSignDefine 4, 4001, ROB_1, 2, 0;
```

Die Testsignalgeschwindigkeit wird verwendet, um die tatsächliche Geschwindigkeit von Achse `2` am Roboter abzulesen.

Vordefinierte Daten

Die folgenden Testsignale stehen für zusätzliche Achsen zur Verfügung und sind im System vordefiniert. Alle Daten sind in SI-Einheiten angegeben und werden an der Motorseite der Achse gemessen.

Symbolische Konstante	Wert	Einheit
`speed`	6	rad/s
`torque_ref`	9	Nm
`resolver_angle`	1	rad
`speed_ref`	4	rad/s
`dig_input1`	102	0 oder 1
`dig_input2`	103	0 oder 1

Die folgenden Testsignale stehen sowohl für Roboter als auch für zusätzliche Achsen zur Verfügung und sind im System vordefiniert. Alle Daten werden an der Armseite der Achse gemessen.

Prüfsignal	Wert	Einheit
Stelle	4000	Grad oder mm[i]
Geschwindigkeit	4001	Grad/Sekunden oder mm/s[i]
Drehmoment	4002	Nm
Externes Drehmoment[ii]	4003	Nm

[i] Welche Einheit verwendet wird, hängt davon ab, ob die Achse rotierend oder linear ist.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.86 testsignal – Prüfsignal

RobotWare - OS

Fortsetzung

ii Liefert ein geschätztes extern angewandtes Drehmoment (durch Kontakt mit der Umgebung). Für eine zusätzliche Achse ist das externe Drehmoment möglicherweise nicht gültig.

Eigenschaften

`testsignal` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Definieren eines Prüfsignals	*TestSignDefine - Definieren eines Prüfsignals auf Seite 881*
Lesen eines Prüfsignals	*TestSignRead - Lesen eines Prüfsignalwerts auf Seite 1554*
Zurücksetzen eines Prüfsignals	*TestSignReset - Zurücksetzen aller Prüfsignaldefinitionen auf Seite 883*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.87 tooldata – Werkzeugdaten

Verwendung

`tooldata` wird verwendet, um die Eigenschaften eines Werkzeugs, z. B. eines Schweißgeräts oder eines Greifers, zu beschreiben. Diese Eigenschaften sind Position und Orientierung des Werkzeugarbeitspunktes (TCP) und die physikalischen Eigenschaften der Werkzeuglast.

Wenn das Werkzeug fest montiert ist (ein stationäres Werkzeug), definieren die Werkzeugdaten zuerst die Position und Orientierung dieses Werkzeugs im Raum, TCP. Dann wird die Last des Greifers beschrieben, der vom Roboter bewegt wird.

Beschreibung

Werkzeugdaten beeinflussen die Roboterbewegungen wie folgt:

- Der Werkzeugarbeitspunkt (TCP) verweist auf einen Punkt, der die Bedingungen für die angegebene Bahn und Geschwindigkeitsleistung erfüllt. Wenn das Werkzeug neuorientiert wird oder koordinierte externe Achsen verwendet werden, folgt nur dieser Punkt der gewünschten Bahn in der programmierten Geschwindigkeit.
- Wenn ein stationäres Werkzeug verwendet wird, beziehen sich programmierte Geschwindigkeit und Bahn auf das Werkobjekt, das der Roboter hält.
- Programmierte Positionen verweisen auf die Position des aktuellen TCP und die Orientierung in Relation zum Werkzeug-Koordinatensystem. Das bedeutet z. B., dass beim Ersetzen eines beschädigten Werkzeugs das alte Programm weiter verwendbar ist, wenn das Werkzeug-Koordinatensystem neu definiert wird.

Werkzeugdaten werden auch beim manuellen Bewegen des Roboters für Folgendes verwendet:

- Definieren des TCP, der sich bei einer Umorientierung des Werkzeugs nicht bewegen soll.
- Definieren des Werkzeug-Koordinatensystems, um die Bewegung und die Rotation in den Werkzeugrichtungen zu vereinfachen.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.87 tooldata – Werkzeugdaten
RobotWare - OS
Fortsetzung

WARNUNG

Die korrekte aktuelle Werkzeuglast sowie, falls vorhanden, die Nutzlast des Roboters müssen unbedingt definiert werden. Falsche Definitionen der Lastdaten können zu Überlastung der mechanischen Roboterstruktur führen. Es besteht zudem ein Risiko, dass die Geschwindigkeit im manuell reduzierten Geschwindigkeitsmodus überschritten werden kann.

Die Angabe falscher Lastdaten kann folgende Konsequenzen haben:

- Der Roboter verwendet möglicherweise nicht seine maximale Kapazität.
- Die Bahngenauigkeit wird beeinträchtigt und es besteht das Risiko des Überschwingens.
- Die mechanische Struktur kann überlastet werden.

Die Steuerung überwacht ununterbrochen die Last und schreibt ein Ereignisprotokoll, wenn die Last höher als erwartet ist. Dieses Ereignisprotokoll wird gespeichert und Steuerungsspeicher protokolliert.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `tooldata`:

Beispiel 1:

```
PERS tooldata gripper := [ TRUE, [[97.4, 0, 223.1], [0.924, 0,
     0.383 ,0]], [5, [23, 0, 75], [1, 0, 0, 0], 0, 0, 0]];
```

Das Werkzeug wird mit den folgenden Werten beschrieben:

- Der Roboter hält das Werkzeug.
- Der TCP befindet sich an Punkt 223,1 mm geradeaus vom Montageflansch und 97,4 mm entlang der X-Achse des Handgelenk-Koordinatensystems.
- Die X'- und Z'-Richtung des Werkzeugs wird 45° im Verhältnis zur Y-Richtung um das Handgelenk-Koordinatensystem gedreht.
- Das Werkzeugmasse beträgt 5 kg.
- Der Schwerpunkt befindet sich am Punkt 75 mm geradeaus vom Montageflansch und 23 mm entlang der x-Achse des Handgelenk-Koordinatensystems.
- Die Last kann als Punktlast betrachtet werden, d. h. ohne Trägheitsmoment.

Beispiel 2:

```
gripper.tframe.trans.z := 225.2;
```

Der TCP des Werkzeugs `gripper` wird um 225.2 in z-Richtung korrigiert.

Vordefinierte Daten

Das Werkzeug `tool0` definiert das Handgelenk-Koordinatensystem mit dem Ursprung als Mittelpunkt des Befestigungsflanschs. Auf `tool0` kann aus dem Programm immer zugegriffen werden, es kann jedoch niemals geändert werden (es ist im Systemmodul BASE gespeichert).

```
PERS tooldata tool0 := [ TRUE, [ [0, 0, 0], [1, 0, 0 ,0] ], [0.001,
     [0, 0, 0.001], [1, 0, 0, 0], 0, 0, 0] ];
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Komponenten

`robhold`

robot hold

Datentyp: `bool`

Definiert, ob der Roboter das Werkzeug hält:

- `TRUE`: **Der Roboter hält das Werkzeug.**
- `FALSE`: **Der Roboter hält das Werkzeug nicht, d. h., ein stationäres Werkzeug wird verwendet.**

`tframe`

tool frame

Datentyp: `pose`

Das Werkzeug-Koordinatensystem, d. h.:

- **Die Position des TCP (x, y und z) in mm, ausgedrückt im Handgelenk-Koordinatensystem (tool0) (siehe folgende Abbildung).**
- **Die Orientierung des Werkzeug-Koordinatensystems, im Handgelenk-Koordinatensystem ausgedrückt (siehe folgende Abbildung).**

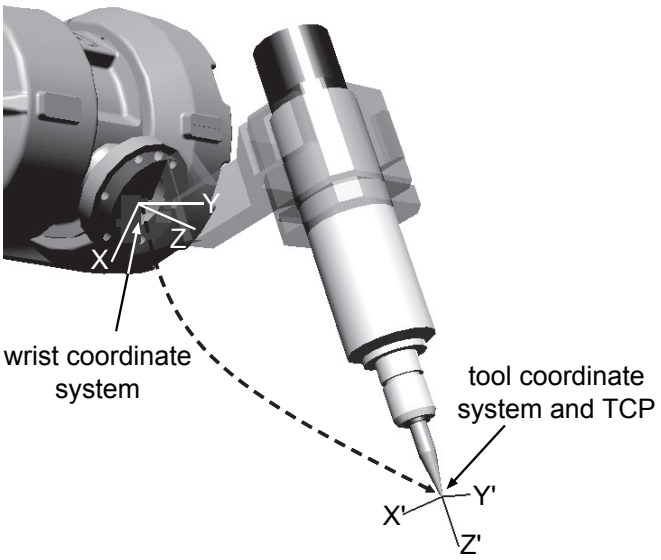


Figure 3.3: Vom Roboter gehaltenes Werkzeug

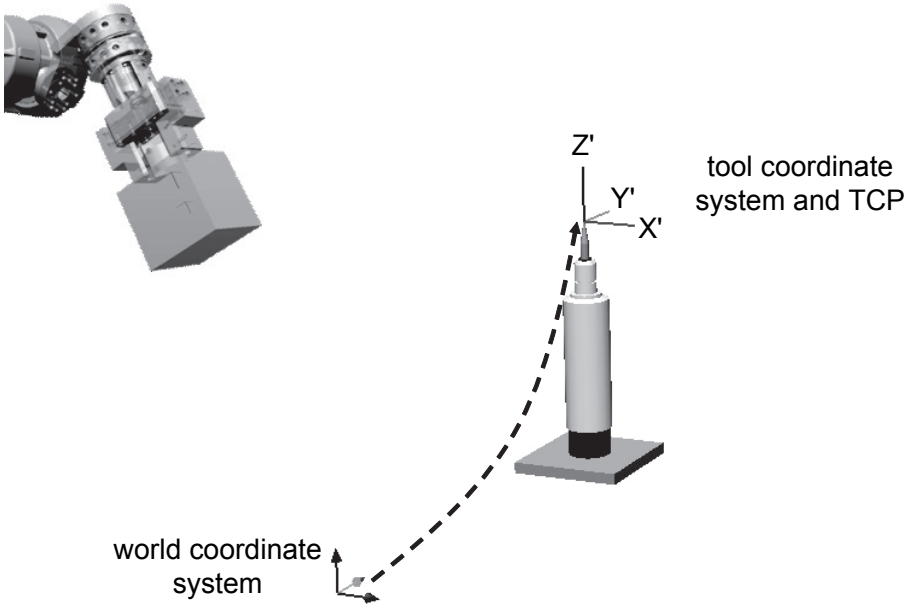
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.87 tooldata – Werkzeugdaten
RobotWare - OS
Fortsetzung

Hinweis

Bei Verwendung eines stationären Werkzeugs erfolgt die Definition des Werkzeug-Koordinatensystems in Relation zum Welt-Koordinatensystem.



xx1100000518

Figure 3.4: Stationäres Werkzeug

`tload`

tool load

Datentyp: `loaddata`

Hinweis

Diese Daten werden für das vom Roboter gehaltene und für das stationäre Werkzeug verwendet. Für das vom Roboter gehaltene Werkzeug beschreiben die Daten die Werkzeuglast. Für ein stationäres Werkzeug beschreiben die Daten die Last des vom Roboter gehaltenen Greifers.

Vom Roboter gehaltenes Werkzeug:

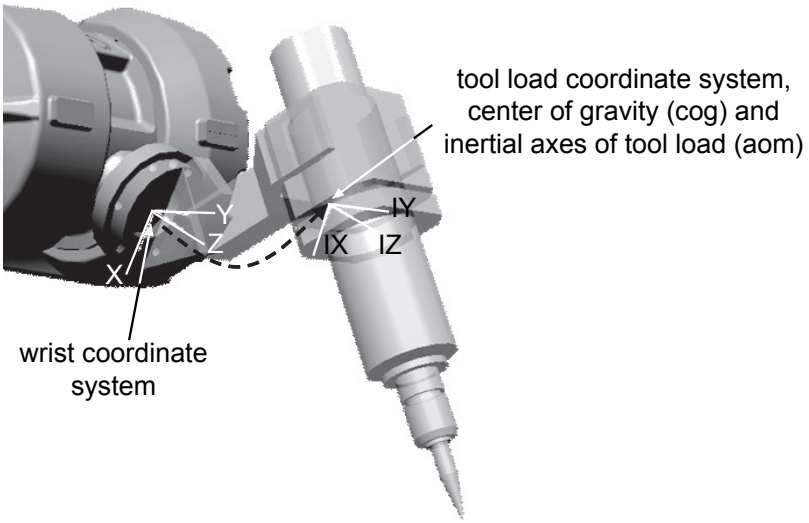
Die Last des Werkzeugs, d. h.:

- Die Masse (Gewicht) des Werkzeugs in kg.
- Der Schwerpunkt der Werkzeuglast (x, y und z) in mm, ausgedrückt im Handgelenk-Koordinatensystem.
- Die Orientierung der Hauptdrehmomentachsen des Werkzeugs, ausgedrückt im Handgelenk-Koordinatensystem

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Die Trägheitsmomente um die Trägheitsmomentachsen in kgm^2. Wenn alle Trägheitskomponenten als 0 kgm^2 sind, wird das Werkzeug als Punktlast behandelt.



xx1100000519

Stationäres Werkzeug:

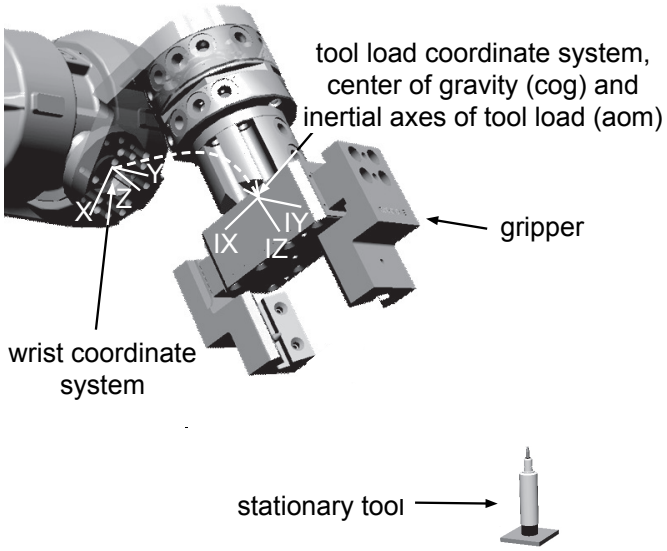
Die Last des Greifers, der das Werkobjekt hält:

- Die Masse (Gewicht) des bewegten Greifers in kg
- Der Schwerpunkt des bewegten Greifers (x, y und z) in mm, ausgedrückt im Handgelenk-Koordinatensystem
- Die Orientierung der Hauptdrehmomentachsen des bewegten Greifers, ausgedrückt im Handgelenk-Koordinatensystem
- Die Trägheitsmomente um die Trägheitsmomentachsen in kgm^2. Wenn alle Trägheitskomponenten als 0 kgm^2 sind, wird der Greifer als Punktlast behandelt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.87 tooldata – Werkzeugdaten
RobotWare - OS
Fortsetzung



xx1100000520

Hinweis

Nur die Last des Werkzeugs/Greifers darf in `tooldata` angegeben werden. Die Nutzlast, die der Greifer handhabt, wird mit der Instruktion `GripLoad` aufgenommen und abgelegt und durch `loaddata` definiert.

Statt der Verwendung der Instruktion `GripLoad` können unterschiedliche `tooldata` für *Greifer mit gegriffenem Werkstück* und *Greifer ohne Werkstück* definiert und verwendet werden.

Zusammenfassung

Die Position und Orientierung von TCP in `tooldata` sind im Handgelenk-Koordinatensystem für ein vom Roboter gehaltenes Werkzeug ausgedrückt.

Die Position und Orientierung von TCP in `tooldata` sind im Welt-Koordinatensystem für ein stationäres Werkzeug ausgedrückt.

Der Teil `loaddata` in `tooldata` steht in allen Fällen mit dem Handgelenk-Koordinatensystem in Verbindung, unabhängig davon, ob ein vom Roboter gehaltenes Werkzeug (zur Beschreibung des Werkzeugs) oder ein stationäres Werkzeug (zur Beschreibung des Greifers) verwendet wird.

Struktur

```
< dataobject of tooldata >
  < robhold of bool >
  < tframe of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
  < tload of loaddata >
    < mass of num >
    < cog of pos >
      < x of num >
      < y of num >
      < z of num >
    < aom of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
    < ix of num >
    < iy of num >
    < iz of num >
```

Einschränkungen

Die Werkzeugdaten dürfen nur als persistente Variable (PERS) und nicht innerhalb einer Routine definiert werden. Die aktuellen Werte werden dann gespeichert, wenn Sie das Programm speichern, und werden beim Laden wieder abgerufen.

Argumente des Typs tooldata in einer beliebigen Bewegungsinstruktion sollten nur eine vollständige Persistente sein (kein Datenfeldelement und keine Datensatzkomponente).

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Coordinate systems*
Definieren der Nutzlast für Roboter	*GripLoad - Definieren der Nutzlast des Roboters auf Seite 242*
Definition der Lastdaten	*loaddata - Lastdaten auf Seite 1723*
Definition von Werkobjekten	*wobjdata - Werkobjektdaten auf Seite 1852*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.88 tpnum - Fensternummer des FlexPendant

Verwendung

`tpnum` stellt die Fensternummer des FlexPendant mit einer symbolischen Konstanten dar.

Beschreibung

Eine `tpnum`-Konstante ist zur Verwendung in der Instruktion `TPShow` vorgesehen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `tpnum`:

Beispiel 1:

```
TPShow TP_LATEST;
```

Das zuletzt am FlexPendant verwendete Fenster vor dem aktuellen Fenster ist nach Abarbeiten dieser Instruktion aktiv.

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `tpnum` sind vordefiniert und können in der Instruktion `TPShow` verwendet werden.

Wert	Symbolische Konstante	Kommentar
2	TP_LATEST	Zuletzt verwendetes Fenster des FlexPendant

Eigenschaften

`tpnum` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Informationen über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic Characteristics - Data Types*
Kommunikation mithilfe des FlexPendant	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID Summary - Communication*
Wechsel des Fensters am FlexPendant	*TPShow - Wechsel des Fensters am FlexPendant auf Seite 900*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.89 trapdata – Interrupt-Daten für aktuellen TRAP

Verwendung

`trapdata` (*trap data*) enthält die Interrupt-Daten, die die Abarbeitung der aktuellen `TRAP`-Routine ausgelöst haben.

Wird in `TRAP`-Routinen verwendet, die durch die Instruktion `IError` vor Verwendung der Instruktion `ReadErrData` erzeugt wurden.

Beschreibung

Daten des Typs `trapdata` stellen interne Informationen zu dem Interrupt dar, der die Abarbeitung der aktuellen Interrupt-Routine (TRAP) ausgelöst hat. Ihr Inhalt hängt vom Typ des Interrupts ab.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `trapdata`:

Beispiel 1:

```
VAR errdomain err_domain;
VAR num err_number;
VAR errtype err_type;
VAR trapdata err_data;
...
TRAP trap_err
  GetTrapData err_data;
  ReadErrData err_data, err_domain, err_number, err_type;
ENDTRAP
```

Wenn ein Fehler in der Interrupt-Routine `trap_err` abgefangen wird, werden Fehlerdomäne, Fehlernummer und Fehlertyp in geeigneten Non-Value-Variablen des Typs `trapdata` gespeichert.

Eigenschaften

`trapdata` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Zusammenfassung von Interrupts	*Technisches Referenzhandbuch - RAPID Overview*
Weitere Informationen zur Interrupt-Verwaltung	*Technisches Referenzhandbuch - RAPID Overview*
Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*
Anfordern eines Interrupts bei Fehlern	*IError - Anfordern eines Interrupts bei Fehlern auf Seite 255*
Abrufen von Interrupt-Daten für aktuellen Interrupt	*GetTrapData - Abrufen von Interrupt-Daten für aktuellen Interrupt auf Seite 238*
Abrufen von Informationen über einen Fehler	*ReadErrData - Abrufen von Informationen über einen Fehler auf Seite 590*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.89 trapdata – Interrupt-Daten für aktuellen TRAP

RobotWare - OS

Fortsetzung

Information über	Siehe
Advanced RAPID	*Produktspezifikation - Steuerungssoftware IRC5*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.90 triggdata – Positioniereereignisse, trigg

Verwendung

`triggdata` wird verwendet, um Daten über ein Positioniereereignis bei einer Roboterbewegung zu speichern.

Ein Positioniereereignis kann das Setzen eines Ausgangssignals oder das Ausführen einer Interrupt-Routine an einer bestimmten Position entlang der Bewegungsbahn des Roboters darstellen.

Beschreibung

Um die Bedingungen für die jeweiligen Maßnahmen bei einem Positioniereereignis zu definieren, werden Variablen des Typs `triggdata` benutzt. Der Dateninhalt der Variable wird im Programm über die Instruktionen `TriggIO`, `TriggEquip`, `TriggCheckIO`, `TriggInt`, `TriggSpeed` oder `TriggRampAO` festgelegt und von einer der Instruktionen `TriggL`, `TriggC` oder `TriggJ` verwendet.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `triggdata` :

Beispiel 1:

```
VAR triggdata gunoff;
TriggIO gunoff, 0,5 \DOp:=gun, 0;
TriggL p1, v500, gunoff, fine, gun1;
```

Das digitale Ausgangssignal `gun` wird auf den Wert `0` gesetzt, wenn sich der TCP an der Position `0,5` mm vor dem Punkt `p1` befindet.

Eigenschaften

`triggdata` ist ein Datentyp ohne einen Wert (Non-Value).

Weitere Informationen

Information über	Siehe
Definition von Triggern	*TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stoppunkt auf Seite 935* *TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930*
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.90 triggdata – Positioniereereignisse, trigg

RobotWare - OS

Fortsetzung

Information über	Siehe
Eigenschaften von Datentypen ohne einen Wert	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.91 triggios – Positioniererereignisse, trigg

Verwendung

`triggios` wird verwendet, um Daten über ein Positioniererereignis bei einer Roboterbewegung zu speichern. Wenn das Positioniererereignis an einer bestimmten Position auf der Bahn erfolgt, wird ein Ausgangssignal auf einen angegeben Wert gesetzt.

Beschreibung

`triggios` wird verwendet, um Bedingungen und Aktionen für das Setzen eines digitalen Ausgangssignals, einer Gruppe digitaler Ausgangssignale oder eines analogen Ausgangssignals an einer festen Position auf der Bewegungsbahn des Roboters zu definieren.

Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `triggios`:

Beispiel 1:

```
VAR triggios gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=1;

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData1:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das Signal `gun` wird gesetzt, wenn sich der TCP 3 mm hinter dem Punkt `p1` befindet.

Komponenten

`used`

Datentyp: `bool`

Legt fest, ob die Datenfeldkomponente verwendet werden soll.

`distance`

Datentyp: `num`

Definiert die Position auf der Bahn, an der das E/A-Ereignis stattfinden soll. Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn, wenn die Komponente `start` auf FALSE gesetzt wurde.

`start`

Datentyp: `bool`

Auf TRUE gesetzt, wenn der Abstand ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.91 triggios – Positioniereregnisse, trigg
RobotWare - OS
Fortsetzung

`equiplag`

Equipment Lag

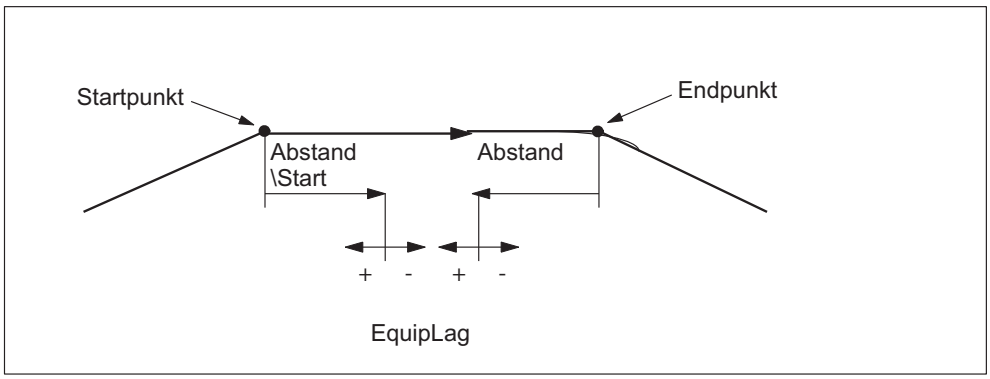
Datentyp: `num`

Gibt die Verzögerung der externen Einrichtung in s an.

Verwenden Sie zur Kompensierung der Verzögerung einen positiven Argumentwert. Ein positiver Wert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, bevor der TCP physisch den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung erreicht hat.

Ein negativer Wert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, nachdem der TCP den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung physisch passiert hat.

Die Abbildung zeigt die Verwendung der Komponente `equiplag`.



xx0800000173

`signalname`

Datentyp: `string`

Der Name des zu ändernden Signals. Es muss sich um ein digitales Ausgangssignal, eine Gruppe von digitalen Ausgangssignalen oder ein analoges Ausgangssignal handeln.

`setvalue`

Datentyp: `num`

Gewünschter Wert des Ausgangssignals (innerhalb des zulässigen Bereichs für das aktuelle Signal).

`xxx`

Datentyp: `num`

Die Komponente wird derzeit nicht verwendet. Sie wird für zusätzliche Funktionalität in zukünftigen Versionen hinzugefügt, wobei die Kompatibilität erhalten bleiben soll.

Struktur

```
<dataobject of triggios>
  <used of bool>
  <distance of num>
  <start of bool>
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
<equiplag of num>
<signalname of string>
<setvalue of num>
<xxx of num>
```

Weitere Informationen

Information über	Siehe
Positioniereereignisse, trigg	*triggiosdnum – Positioniereereignisse, trigg auf Seite 1838*
Lineare Roboterbewegungen mit E/A-Ereignissen	*TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen auf Seite 967*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.92 triggiosdnum – Positioniereereignisse, trigg

Verwendung

`triggiosdnum` wird verwendet, um Daten über ein Positioniereereignis bei einer Roboterbewegung zu speichern. Wenn das Positioniereereignis an einer bestimmten Position auf der Bahn erfolgt, wird ein Ausgangssignal auf einen angegeben Wert gesetzt.

Beschreibung

`triggiosdnum` wird verwendet, um Bedingungen und Aktionen für das Setzen eines digitalen Ausgangssignals, einer Gruppe digitaler Ausgangssignale oder eines analogen Ausgangssignals an einer festen Position auf der Bewegungsbahn des Roboters zu definieren.

Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `triggiosdnum`:

Beispiel 1:

```
VAR triggiosdnum gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="go_gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:=123456789;

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData3:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Das Signal `go_gun` wird gesetzt, wenn sich der TCP 3 mm hinter dem Punkt `p1` befindet.

Komponenten

`used`

Datentyp: `bool`

Legt fest, ob die Datenfeldkomponente verwendet werden soll.

`distance`

Datentyp: `num`

Definiert die Position auf der Bahn, an der das E/A-Ereignis stattfinden soll. Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn, wenn die Komponente `start` auf FALSE gesetzt wurde.

`start`

Datentyp: `bool`

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Auf TRUE gesetzt, wenn der Abstand ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`equiplag`

Equipment Lag

Datentyp: `num`

Gibt die Verzögerung der externen Einrichtung in s an.

Verwenden Sie zur Kompensierung der Verzögerung einen positiven Argumentwert. Ein positiver Wert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, bevor der TCP physisch den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung erreicht hat.

Ein negativer Wert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, nachdem der TCP den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung physisch passiert hat.

`signalname`

Datentyp: `string`

Der Name des zu ändernden Signals. Es muss sich um ein digitales Ausgangssignal, eine Gruppe von digitalen Ausgangssignalen oder ein analoges Ausgangssignal handeln.

`setvalue`

Datentyp: `dnum`

Gewünschter Wert des Ausgangssignals (innerhalb des zulässigen Bereichs für das aktuelle Signal).

`xxx`

Datentyp: `num`

Die Komponente wird derzeit nicht verwendet. Sie wird für zusätzliche Funktionalität in zukünftigen Versionen hinzugefügt, wobei die Kompatibilität erhalten bleiben soll.

Struktur

```
<dataobject of triggiosdnum>
  <used of bool>
  <distance of num>
  <start of bool>
  <equiplag of num>
  <signalname of string>
  <setvalue of dnum>
  <xxx of num>
```

Weitere Informationen

Information über	Siehe
Positioniereereignisse, trigg	*triggios – Positioniereereignisse, trigg auf Seite 1835*
Lineare Roboterbewegungen mit E/A-Ereignissen	*TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen auf Seite 967*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.93 triggmode - Triggeraktionsmodus

Verwendung

`triggmode` wird verwendet, um verschiedene Aktionsmodi bei der Trigger-Definition anzugeben.

Beschreibung

Eine konstanter `triggmode` soll verwendet werden, um den Modus für Instruktionen zur Definition von Triggern festzulegen.

Grundlegende Beispiele

Die folgenden Beispiele veranschaulichen den Datentyp `triggmode`:

Beispiel 1:

```
CONNECT intno1 WITH trap1;
TriggInt trigg1, Distance:=17, intno1 \Inhib:=inhibit
     \Mode:=TRIGG_MODE1;
TriggL p1, v500, trigg1, z50, gun1;
```

Die Abbruchroutine `trap1` wird ausgeführt, wenn das TCP sich in einer Position von17 mm vor dem Punkt`p1` befindet, und die persistente Variable eine `inhibit` Markierung ist `TRUE` (Modus `TRIGG_MODE1` kehrt den Wert um, der von der `inhibit` Markierung abgelesen wird).

Beispiel 2:

```
TriggEquip trigg1, 17, 0 \GOp:=go1, SetValue:=5 \Inhib:=inhibit
     \Mode:=TRIGG_MODE2;
TriggL p1, v500, trigg1, z50, gun1;
```

Wenn das persistente Flag `inhibit` `FALSE` ist, wenn der TCP an einer Position 17 mm vor dem Punkt `p1`ist, wird das E/A-Signal `go1` auf den in `SetValue` angegebenen Wert festgelegt. Ist die persistente Variable `inhibit` gleich `TRUE`, wird überhaupt keine Aktion ausgeführt (Wert des E/A-Signals `go1` beibehalten).

Beispiel 3

```
TriggEquip trigg1, 17, 0 \GOp:=go1, SetValue:=0 \Inhib:=inhibit
     \InhibSetValue:=setDnum \Mode:=TRIGG_MODE3;
TriggL p1, v500, trigg1, z50, gun1;
```

Wenn das persistente Flag `inhibit` `TRUE` ist, wenn der TCP an einer Position 17 mm vor dem Punkt `p1` ist, wird das E/A-Signal `go1` auf den Wert gesetzt, der aus der `dnum` persistenten Variable `setDnum` ausgelesen wird. Wenn `inhibit` `FALSE` ist, wird überhaupt keine Aktion ausgeführt (Wert des E/A-Signals `go1` beibehalten).

Beispiel 4:

```
TriggEquip trigg1, 17, 0 \GOp:=go1, SetValue:=5 \Inhib:=inhibit
     \Mode:=TRIGG_MODE3;
TriggL p1, v500, trigg1, z50, gun1;
```

Wenn das persistente Flag `inhibit` `TRUE` lautet, wenn sich der TCP in einer Position 17 mm vor dem Punkt `p1` befindet, dann wird das E/A-Signal `go1` auf den Wert gesetzt, der in `SetValue` angegeben ist (in diesem Beispiel 5). Wenn `inhibit` `FALSE` lautet, erfolgt keinerlei Aktion (der Wert des E/A-Signals `go1` bleibt erhalten).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `triggmode` sind vordefiniert und können verwendet werden, um den Aktionsmodus bei der Definition der Trigger zu spezifizieren.

Wert	Symbolische Konstante	Kommentar
1	`TRIGG_MODE1`	Kann von den Instruktionen `TriggCheckIO`, `TriggEquip`, `TriggIO`, `TriggInt`, `TriggSpeed` und `TriggRampAO` verwendet werden. Gelesenen Wert der persistenten Variable im optionalen Argument `Inhib` umkehren.
2	`TRIGG_MODE2`	Kann von den Instruktionen `TriggEquip`, `TriggIO`, `TriggSpeed` und `TriggRampAO` verwendet werden, wenn das optionale Argument `Inhib` verwendet wird. Wenn der tatsächliche Wert des angegebenen Flags, das in `Inhib` verwendet wird, an der Positionszeit für die Einstellung des E/A-Signals `TRUE` ist, dann wird das angegebene E/A-Signal nicht aktualisiert (keine Aktion).
3	`TRIGG_MODE3`	Der Modus kann nur zusammen mit optionalen Argument `Inhib` in den Instruktionen `TriggEquip` und `TriggIO` verwendet werden. Wenn der aktuelle Wert des angegebenen Flags, das in `Inhib` verwendet wird, `FALSE` ist, wird das E/A-Signal nicht aktualisiert (keine Aktion). Wenn die Verwendung nur mit `Inhib` erfolgt und das in `Inhib` angegebene Flag an der Position `TRUE` ist, wird das E/A-Signalt auf einen Wert gesetzt, der in Argument `SetValue` oder `SetDvalue` angegeben ist. Verwendung mit `Inhib` und mit `InhibSetValue`: Die Argumente `SetValue` und `SetDvalue` werden nicht berücksichtigt. Wenn der tatsächliche Wert des angegebenen Flags, das in `Inhib` verwendet wird, `TRUE` ist, dann wird das E/A-Signal auf den Wert gesetzt, den die persistente Variable an der Position besitzt, die im optionalen Argument `InhibSetValue` verwendet wird.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.93 triggmode - Triggeraktionsmodus
RobotWare - OS
Fortsetzung

Eigenschaften

`triggmode` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Verwenden von Triggern	*TriggL - Lineare Roboterbewegungen mit Ereignissen auf Seite 950* *TriggC - Kreisförmige Roboterbewegung mit Ereignissen auf Seite 904* *TriggJ - Achsenweise Roboterbewegungen mit Ereignissen auf Seite 942*
Definition von Triggern	*TriggEquip - Definieren eines positions- und zeitabhängigen E/A-Ereignisses auf der Bahn auf Seite 923* *TriggInt - Definieren eines positionsabhängigen Interrupts auf Seite 930* *TriggIO - Definieren eines positions- oder zeitabhängigen E/A-Ereignisses nahe einem Stopppunkt auf Seite 935* *TriggRampAO - Definieren eines AO-Rampenereignisses an einer festen Position auf der Bahn auf Seite 975* *TriggSpeed - Definition der analogen, geschwindigkeitsproportionalen Ausgangssignale mit positions- und zeitabhängiger Skalierung auf Seite 983*
Definieren einer positionsabhängigen E/A-Prüfung	*TriggCheckIO - Definieren einer E/A-Prüfung an einer festen Position auf Seite 913*
Speichern von Triggerdaten	*triggdata – Positioniereereignisse, trigg auf Seite 1833*
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.94 triggstrgo - Positioning events, trigg

Verwendung

`triggstrgo`*(trigg stringdig group output)* wird verwendet, um Daten über ein Positioniereereignis bei einer Roboterbewegung zu speichern. Wenn das Positioniereereignis an einer bestimmten Position auf der Bahn erfolgt, wird eine Gruppe von digitalen Ausgangssignalen auf einen angegebenen Wert gesetzt.

Beschreibung

`triggstrgo` wird verwendet, um Bedingungen und Aktionen für das Setzen einer Gruppe von digitalen Ausgangssignalen an einer festen Position auf der Bewegungsbahn des Roboters zu definieren.

Komponenten

`used`

Datentyp: `bool`

Legt fest, ob die Datenfeldkomponente verwendet werden soll.

`distance`

Datentyp: `num`

Definiert die Position auf der Bahn, an der das E/A-Ereignis stattfinden soll. Angegeben als Abstand in mm (positiver Wert) vom Endpunkt der Bewegungsbahn, wenn die Komponente `start` auf FALSE gesetzt wurde.

`start`

Datentyp: `bool`

Auf TRUE gesetzt, wenn der Abstand ab dem Startpunkt anstelle des Endpunkts der Bewegung gemessen wird.

`equiplag`

Equipment Lag

Datentyp: `num`

Gibt die Verzögerung der externen Einrichtung in s an.

Verwenden Sie zur Kompensierung der Verzögerung einen positiven Argumentwert. Ein positiver Wert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, bevor der TCP physisch den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung erreicht hat.

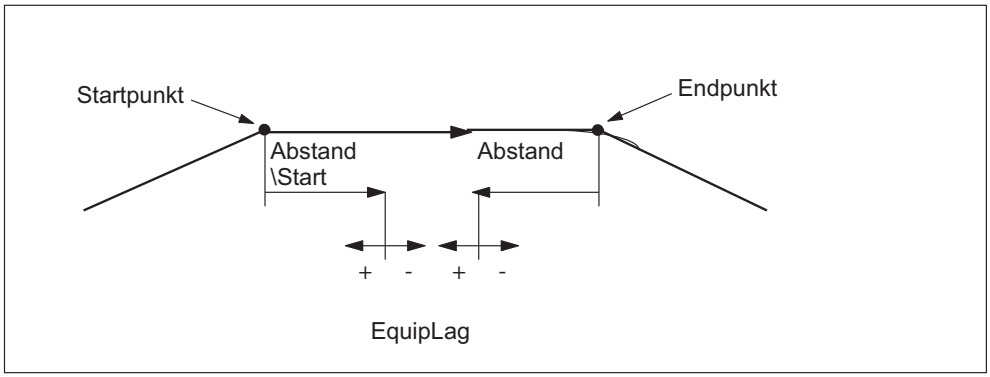
Ein negativer Wert bedeutet, dass das Robotersystem das E/A-Signal zur angegebenen Zeit setzt, nachdem der TCP den angegebenen Abstand in Relation zum Start- oder Endpunkt der Bewegung physisch passiert hat.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.94 triggstrgo - Positioning events, trigg
RobotWare - OS
Fortsetzung

Die Abbildung zeigt die Verwendung der Komponente `equiplag`.



xx0800000173

`signalname`

Datentyp: `string`

Der Name des zu ändernden Signals. Dies muss der Name eines Gruppenausgangssignals sein.

`setvalue`

Datentyp: `stringdig`

Gewünschter Wert des Ausgangssignals (innerhalb des zulässigen Bereichs für das aktuelle digitale Gruppenausgangssignal). Der Datentyp `stringdig` ermöglicht die Verwendung von Werten bis 4294967295. Dies ist der maximal zulässige Wert für eine Gruppe digitaler Signale (ein Gruppensignal im System darf maximal 32 Signale enthalten).

`xxx`

Datentyp: `num`

Die Komponente wird derzeit nicht verwendet. Sie wird für zusätzliche Funktionalität in zukünftigen Versionen hinzugefügt, wobei die Kompatibilität erhalten bleiben soll.

Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `triggstrgo`:

Beispiel 1:

```
VAR triggstrgo gunon{1};

gunon{1}.used:=TRUE;
gunon{1}.distance:=3;
gunon{1}.start:=TRUE;
gunon{1}.signalname:="gun";
gunon{1}.equiplag:=0;
gunon{1}.setvalue:="4294967295";

MoveJ p1, v500, z50, gun1;
TriggLIOs p2, v500, \TriggData2:=gunon, z50, gun1;
MoveL p3, v500, z50, gun1;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Das Signal `gun` wird auf den Wert 4294967295 gesetzt, wenn sich der TCP 3 mm hinter dem Punkt `p1` befindet.

Struktur

```
<dataobject of triggstrgo>
  <used of bool>
  <distance of num>
  <start of bool>
  <equiplag of num>
  <signalname of string>
  <setvalue of stringdig>
  <xxx of num>
```

Weitere Informationen

Information über	Siehe
Lineare Roboterbewegungen mit E/A-Ereignissen	*TriggLIOs - lineare Roboterbewegungen mit E/A-Ereignissen auf Seite 967*
Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten	*StrDigCmp - Vergleichen von zwei Zeichenfolgen, die nur Ziffern enthalten auf Seite 1518*
Arithmetische Operationen am Datentyp stringdig	*StrDigCalc - Arithmetische Operationen mit dem Datentyp stringdig auf Seite 1515*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.95 tsp_status – Status des Taskauswahlfeldes

Verwendung

`tsp_status` wird verwendet, um den Status des **Taskauswahlfeldes** auf dem FlexPendant widerzuspiegeln.

Beschreibung

Mit den Funktionen `TaskIsActive` und `GetTSPStatus` ist es möglich, den aktuellen Status des **Taskauswahlfeldes** auf dem FlexPendant abzulesen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `tsp_status`:

Beispiel 1:

```
VAR tsp_status tspstatus;
...
tspstatus:=GetTSPStatus("MYTASK");
IF tspstatus >= TSP_NORMAL_UNCHECKED AND tspstatus <=
     TSP_SEMISTATIC_UNCHECKED THEN
  TPWrite "Task MYTASK is unchecked in the Task Selection Panel";
ELSEIF tspstatus >= TSP_NORMAL_CHECKED THEN
  TPWrite "Task MYTASK is checked in the Task Selection Panel";
ELSE
  TPWrite "Task MYTASK is unchecked in TSP due to execution in
       service routine";
ENDIF
```

In diesem Programmbeispiel wird untersucht, ob die Programmtask `MYTASK` im **Taskauswahlfeld** auf dem FlexPendant ausgewählt oder abgewählt ist.

Vordefinierte Daten

Folgende Konstanten des Typs `tsp_status` sind vordefiniert:

RAPID Konstante	Wert	Beschreibung
`TSP_STATUS_NOT_NORMAL_TASK`	0	`TaskIsActive`. Die Task ist eine semistatische oder statische Task, keine normale Task.
`TSP_STATUS_DEACT`	1	`TaskIsActive`. Die normale Task ist im Taskauswahlfeld deaktiviert.
`TSP_STATUS_DEACT_SERV_ROUT`	2	`TaskIsActive`. Die normale Task ist im Taskauswahlfeld deaktiviert, weil eine andere Task eine Serviceroutine ausführt.
`TSP_STATUS_ACT`	3	`TaskIsActive`. Die normale Task ist im Taskauswahlfeld aktiv.
`TSP_UNCHECKED_RUN_SERV_ROUT`	10	`GetTSPStatus`. Die normale Task ist im Taskauswahlfeld abgewählt, weil eine andere Task eine Serviceroutine ausführt.
`TSP_NORMAL_UNCHECKED`	11	`GetTSPStatus`. Die normale Task ist im Taskauswahlfeld abgewählt.
`TSP_STATIC_UNCHECKED`	12	`GetTSPStatus`. Die statische Task ist im Taskauswahlfeld abgewählt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

RAPID Konstante	Wert	Beschreibung
TSP_SEMISTATIC_UNCHECKED	13	GetTSPStatus. Die semistatische Task ist im Taskauswahlfeld abgewählt.
TSP_NORMAL_CHECKED	14	GetTSPStatus. Die normale Task ist im Taskauswahlfeld ausgewählt.
TSP_STATIC_CHECKED	15	GetTSPStatus. Die statische Task ist im Taskauswahlfeld ausgewählt.
TSP_SEMISTATIC_CHECKED	16	GetTSPStatus. Die semistatische Task ist im Taskauswahlfeld ausgewählt.

Eigenschaften

tsp_status ist ein Alias-Datentyp für num und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Rufen Sie den aktuellen Status des Taskauswahlfeldes ab	*GetTSPStatus – Rufen Sie den aktuellen Status des Taskauswahlfeldes ab auf Seite 1344*
Prüfen Sie, ob eine normale Task aktiv ist	*TaskIsActive – Überprüfen Sie, ob eine normale Task aktiv ist auf Seite 1545*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.96 tunetype – Servo-Abstimmungstyp

Verwendung

`tunetype` stellt eine Ganzzahl mit einer symbolischen Konstanten für verschiedene Arten der Servo-Abstimmung dar.

Beschreibung

Eine Tune-Type-Konstante ist als Argument der Instruktion `TuneServo` vorgesehen.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `tunetype`:

Beispiel 1:

```
TuneServo MHA160R1, 1, 110 \Type:= TUNE_KP;
```

Vordefinierte Daten

Die folgenden symbolischen Konstanten des Datentyps `tunetype` sind vordefiniert und können als Argument für die Instruktion `TuneServo` verwendet werden.

Wert	Symbolische Konstante	Kommentar
0	`TUNE_DF`	Verringert Überschwingen
1	`TUNE_KP`	Beeinflusst Positionssteuerung
2	`TUNE_KV`	Beeinflusst Geschwindigkeitssteuerung
3	`TUNE_TI`	Beeinflusst die Integrationsdauer der Geschwindigkeitssteuerung
4	`TUNE_FRIC_LEV`	Beeinflusst den Grad der Reibungskompensierung
5	`TUNE_FRIC_RAMP`	Beeinflusst die Rampe der Reibungskompensierung
6	`TUNE_DG`	Verringert Überschwingen
7	`TUNE_DH`	Reduziert Vibration bei schweren Lasten
8	`TUNE_DI`	Verringert Bahnfehler
9	`TUNE_DK`	Nur für ABB-interne Verwendung
10	`TUNE_DL`	Nur für ABB-interne Verwendung

Eigenschaften

`tunetype` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
Datentypen im Allgemeinen, Alias-Datentypen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Basic characteristics - Data types*
Verwenden des Datentyps "tunetype"	*TuneServo - Abstimmen von Servos auf Seite 1004*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.97 uishownum - Instanz-ID für UIShow

Verwendung

`uishownum` ist der für den Parameter `InstanceId` in der Instruktion `UIShow` verwendete Datentyp. Er dient zum Identifizieren einer Ansicht am FlexPendant.

Beschreibung

Wenn eine persistente Variable vom Typ `uishownum` mit der Instruktion `UIShow` verwendet wird, erhält sie einen bestimmten Wert, der die am FlexPendant gestartete Ansicht identifiziert. Diese persistente Variable wird anschließend in allen Aktionen mit dieser Ansicht verwendet, z. B. erneutes Starten der Ansicht, Ändern der Ansicht usw.

Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `uishownum`:

Beispiel 1:

```
CONST string Name:="TpsViewMyAppl.gtpu.dll";
CONST string Type:="ABB.Robotics.SDK.Views.TpsViewMyAppl";
CONST string Cmd1:="Init data string passed to the view";
PERS uishownum myinstance:=0;
VAR num mystatus:=0;
...
! Launch one view of the application MyAppl
UIShow Name, Type \InitCmd:=Cmd1 \InstanceID:=myinstance
     \Status:=mystatus;
```

Der Code oben startet eine Ansicht der Anwendung `MyAppl` mit dem Initialisierungsbefehl `Cmd1`. Das Token zum Identifizieren der Ansicht wird im Parameter `myinstance` gespeichert.

Eigenschaften

`uishownum` ist ein Alias-Datentyp für `num` und übernimmt daher dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
UIShow	*UIShow - Anzeigen der Benutzeroberfläche auf Seite 1026*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.98 weavestartdata - Pendelstartdaten

Verwendung

`weavestartdata` werden bei der Steuerung der stationären Pendelbewegungen beim Start und Neustart eines Prozesses des CAP verwendet.

`weavestartdata` ist eine Komponente der `capdata` und definiert die Eigenschaften der stationären Pendelbewegung beim Start oder Neustart eines CAP-Prozesses:

- Wenn beim Start stationäre Pendelbewegungen stattfinden sollen (`active`)
- Breite der stationären Pendelbewegung (`width`)
- Richtung relativ zur Bahnrichtung (`dir`)
- Frequenz der stationären Pendelbewegung (`cycle_time`)

Stationäre Pendelbewegungen erfolgen immer geometrisch mit Zick-Zack-Muster, siehe *capweavedata - Pendeldaten für CAP auf Seite 1667*.

Komponenten

active

Datentyp: `bool`

Wert	Beschreibung
TRUE	Stationäre Pendelbewegungen zu Beginn eines CAP-Prozess ausführen
FALSE	Stationäre Pendelbewegungen zu Beginn eines CAP-Prozess NICHT ausführen

width

Datentyp: `num`

Legt die Amplitude der stationären Pendelbewegung fest (mm).

dir

Datentyp: `num`

Definiert die Richtung der stationären Pendelbewegung relativ zur Bahnrichtung (Grad). Null Grad bedeutet senkrechtes Pendeln sowohl in Bezug auf die Bahn als auch auf die z-Koordinate des Werkzeugs.

cycle_time

Datentyp: `num`

Definiert die Gesamtzeit (in Sekunden) eines vollständigen stationären Pendelbewegungszyklus, d. h., die Definition der Pendelfrequenz. Die Pendelbewegung wird bis zum Prozessstart anhalten, d. h., die Überwachungskriterien der Phase START_MAIN sind erfüllt.

Syntax

```
< data object of weavestartdata >
  < active of bool >
  < width of num >
  < dir of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
< cycle_time of num >
```

Weitere Informationen

	Beschrieben in:
`capdata` **Datentyp**	*capdata - CAP-Daten auf Seite 1653*
Continuous Application Platform	*Application manual - Continuous Application Platform*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.99 wobjdata - Werkobjektdaten

Verwendung

`wobjdata` wird verwendet, um das Werkobjekt zu beschreiben, das der Roboter schweißt, verarbeitet, bewegt usw.

Beschreibung

Wenn Werkobjekte in einer Positionierinstruktion definiert werden, beruht die Position auf den Koordinaten des Werkobjekts. Dies hat die folgenden Vorteile:

- Bei manueller Eingabe von Positionsdaten, z. B. in der Offline-Programmierung, können die Werte häufig aus einer Zeichnung übernommen werden.
- Programme können nach Änderungen in der Roboterinstallation schnell wieder verwendet werden. Wenn z. B. eine Vorrichtung verlagert wird, muss nur das Anwender-Koordinatensystem umdefiniert werden.
- Variationen in der Befestigung des Werkobjekts können kompensiert werden. Dafür ist jedoch ein Sensor erforderlich, um das Werkobjekt zu positionieren.

Bei Einsatz eines stationären Werkzeugs oder von koordinierten externen Achsen muss das Werkobjekt definiert werden, da die Bahn und Geschwindigkeit sich dann auf das Werkobjekt anstelle des TCP beziehen.

Werkobjektdaten können auch für die manuelle Bewegung verwendet werden:

- Der Roboter lässt sich in die Richtungen des Werkobjekts manuell bewegen.
- Die aktuelle angezeigte Position beruht auf dem Koordinatensystem des Werkobjekts.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `wobjdata`:

Beispiel 1:

```
PERS wobjdata wobj2 :=[ FALSE, TRUE, "", [ [300, 600, 200], [1, 0,
     0 ,0] ], [ [0, 200, 30], [1, 0, 0 ,0] ] ];
```

Das Werkobjekt in obiger Abbildung wird mit den folgenden Werten beschrieben:

- Der Roboter hält das Werkobjekt nicht.
- Das feste Anwender-Koordinatensystem wird verwendet.
- Das Anwender-Koordinatensystem ist nicht gedreht und die Koordinaten seines Ursprungs sind im Welt-Koordinatensystem x = 300, y = 600 und z = 200 mm.
- Das Objekt-Koordinatensystem ist nicht gedreht und die Koordinaten seines Ursprungs sind im Anwender-Koordinatensystem x= `0`, y= 200 und z = 30 mm.

  ```
  wobj2.oframe.trans.z := 38.3;
  ```

- Die Position des Werkobjekts `wobj2` wird um 38,3 mm in z-Richtung angepasst.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Vordefinierte Daten

Die Werkobjektdaten `wobj0` werden so definiert, dass das Objekt-Koordinatensystem mit dem Welt-Koordinatensystem identisch ist. Der Roboter hält das Werkobjekt nicht.

`Wobj0` zugreifen, aber die Daten lassen sich nicht ändern (sie sind im Systemmodul BASE gespeichert).

```
PERS wobjdata wobj0 := [ FALSE, TRUE, "", [ [0, 0, 0], [1, 0, 0
     ,0] ], [ [0, 0, 0], [1, 0, 0 ,0] ] ];
```

Komponenten

`robhold`

robot hold

Datentyp: `bool`

Definiert, ob der Roboter in der aktuellen Programmtask das Werkobjekt hält:

- `TRUE`: Der Roboter hält das Werkobjekt, d. h., es wird ein stationäres Werkzeug verwendet.
- `FALSE`: Der Roboter hält das Werkobjekt nicht, d. h., er hält das Werkzeug.

`ufprog`

user frame programmed

Datentyp: `bool`

Definiert, ob ein festes Anwender-Koordinatensystem verwendet wird:

- `TRUE`: Festes Anwender-Koordinatensystem
- `FALSE`: Bewegliches Anwender-Koordinatensystem, d. h., koordinierte externe Achsen werden verwendet. Auch zur Verwendung in einem MultiMove-System im teilkoordinierten oder synchronen, koordinierten Bewegungsmodus.

`ufmec`

user frame mechanical unit

Datentyp: `string`

Die mechanische Einheit, mit der die Roboterbewegungen koordiniert werden. Wird nur im Fall von beweglichen Anwender-Koordinatensystemen angegeben (`ufprog` ist `FALSE`).

Gibt den Namen der mechanischen Einheit an, der in den Systemparametern festgelegt ist, z. B. `orbit_a`.

`uframe`

user frame

Datentyp: `pose`

Das Anwender-Koordinatensystem, d. h. die Position der aktuellen Arbeitsfläche oder Vorrichtung (siehe folgende Abbildung):

- Die Position des Ursprungs des Koordinatensystems (x, y und z) in mm
- Die Drehung des Koordinatensystems als Quaternion (q1, q2, q3, q4) ausgedrückt

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.99 wobjdata - Werkobjektdaten
RobotWare - OS
Fortsetzung

Wenn der Roboter das Werkzeug hält, wird das Anwender-Koordinatensystem im Welt-Koordinatensystem definiert (bei Verwendung eines stationären Werkzeugs im Handgelenk-Koordinatensystem).

Für ein bewegliches Anwender-Koordinatensystem (`ufprog` ist `FALSE`) wird das Anwender-Koordinatensystem fortlaufend durch das System definiert.

`oframe`

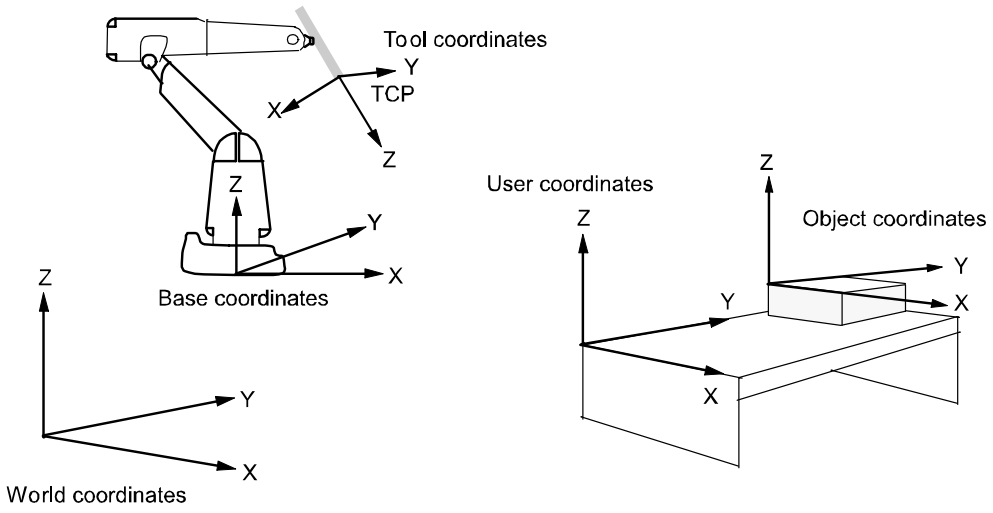
object frame

Datentyp: `pose`

Das Objekt-Koordinatensystem, d. h. die Position des aktuellen Werkobjekts (siehe folgende Abbildung):

- Die Position des Ursprungs des Koordinatensystems (x, y und z) in mm
- Die Drehung des Koordinatensystems als Quaternion (q1, q2, q3, q4) ausgedrückt

Das Objekt-Koordinatensystem wird im Anwender-Koordinatensystem definiert.



xx0500002369

Struktur

```
< dataobject of wobjdata >
  < robhold of bool >
  < ufprog of bool >
  < ufmec of string >
  < uframe of pose >
    < trans of pos >
      < x of num >
      < y of num >
      < z of num >
    < rot of orient >
      < q1 of num >
      < q2 of num >
      < q3 of num >
      < q4 of num >
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
< oframe of pose >
  < trans of pos >
    < x of num >
    < y of num >
    < z of num >
  < rot of orient >
    < q1 of num >
    < q2 of num >
    < q3 of num >
    < q4 of num >
```

Einschränkungen

Die Werkobjektdaten dürfen nur als persistente Variable (PERS) und nicht innerhalb einer Routine definiert werden. Die aktuellen Werte werden dann gespeichert, wenn Sie das Programm speichern, und werden beim Laden wieder abgerufen.

Argumente des Typs wobjdata in einer beliebigen Bewegungsinstruktion sollten nur eine vollständige Persistente sein (kein Datenfeldelement und keine Datensatzkomponente).

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *RAPID summary - Motion*
Koordinatensysteme	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Coordinate systems*
Koordinierte externe Achsen	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O Principles - Coordinate systems*
Kalibrierung koordinierter Achsen	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung* *Anwendungshandbuch - MultiMove*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.100 wzstationary - Stationäre Weltzonendaten

Verwendung

`wzstationary` (*world zone stationary*) dient zum Identifizieren einer stationären Weltzone und kann nur in einer Ereignisroutine verwendet werden, die mit dem Ereignis POWER ON verbunden ist.

Eine Weltzone wird für Roboterbewegungen bei der Programmabarbeitung sowie beim manuellen Bewegen des Roboters überwacht. Wenn der TCP des Roboters oder die Roboter- oder externen Achsen die Weltzone erreichen, wird die Bewegung gestoppt oder ein digitales Ausgangssignal gesetzt oder zurückgesetzt.

Beschreibung

Eine `wzstationary`-Weltzone wird durch die Instruktion `WZLimSup` oder `WZDOSet` definiert und aktiviert.

`WZLimSup` oder `WZDOSet` weist der Variable oder Persistenten des Datentyps `wzstationary` einen numerischen Wert zu. Der Wert identifiziert die Weltzone.

Eine stationäre Weltzone ist im Motoren-EIN-Zustand **immer aktiv** und wird nur durch einen **Neustart** gelöscht. Eine stationäre Weltzone lässt sich nicht über RAPID-Instruktionen deaktivieren, aktivieren oder löschen.

Stationäre Weltzonen sollten ab dem Einschalten aktiv sein und müssen in einer NETZ_EIN-Ereignisroutine oder einer semistatischen Task definiert sein.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `wzstationary`:

Beispiel 1:

```
VAR wzstationary conveyor;
...
PROC ...
  VAR shapedata volume;
  ...
  WZBoxDef \Inside, volume, p_corner1, p_corner2;
  WZLimSup \Stat, conveyor, volume;
ENDPROC
```

Ein `conveyor` (Conveyor) wird als Quader definiert (das Volumen unter dem Förderband). Wenn der Roboter dieses Volumen erreicht, wird die Bewegung gestoppt.

Weitere Beispiele

Ein vollständiges Beispiel finden Sie unter der Instruktion `WZLimSup`.

Einschränkungen

`wzstationary` -Daten können als Variable (**VAR**) oder Persistente (**PERS**) definiert werden. Sie sind global in einer Task oder lokal in einem Modul möglich, jedoch nicht lokal in einer Routine.

Argumente des Typs `wzstationary` dürfen nur vollständige Daten darstellen (nicht ein Datenfeldelement oder eine Datensatzkomponente).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Die Steuerung verwendet keinen Anfangswert für Daten des Typs `wzstationary`. Wenn eine persistente Variable in einem Multitasking-System verwendet werden muss, setzen Sie den Anfangswert in beiden Tasks auf 0, z. B. `PERS`

```
wzstationary share_workarea := [0];
```

Eigenschaften

`wzstationary` ist ein Alias-Datentyp von `wztemporary` und übernimmt dessen Eigenschaften.

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - World zones*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Temporäre Weltzone	*wztemporary - Temporäre Weltzonendaten auf Seite 1858*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.101 wztemporary - Temporäre Weltzonendaten

Verwendung

`wztemporary` (*world zone temporary*) wird verwendet, um eine temporäre Weltzone zu identifizieren, und kann an beliebiger Stelle im RAPID-Programm für jede Bewegungstask verwendet werden.

Eine Weltzone wird für Roboterbewegungen bei der Programmabarbeitung sowie beim manuellen Bewegen des Roboters überwacht. Wenn der TCP des Roboters oder die Roboter- oder externen Achsen die Weltzone erreichen, wird die Bewegung gestoppt oder ein digitales Ausgangssignal gesetzt oder zurückgesetzt.

Beschreibung

Eine `wztemporary`-Weltzone wird durch die Instruktion `WZLimSup` oder `WZDOSet` definiert und aktiviert.

`WZLimSup` oder `WZDOSet` weist der Variable oder Persistenten des Datentyps `wztemporary` einen numerischen Wert zu. Der Wert identifiziert die Weltzone.

Eine definierte und aktivierte Weltzone lässt sich deaktivieren durch `WZDisable`, erneut aktivieren durch `WZEnable` und löschen durch `WZFree`.

Alle temporären Weltzonen in der Bewegungstask werden automatisch gelöscht und alle Datenobjekte vom Typ `wztemporary` in der Bewegungstask werden auf 0 gesetzt:

- beim Laden eines neuen Programms in die motion -Task
- wenn das Programm ab dem Anfang der motion-Task gestartet wird.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `wztemporary`:

Beispiel 1:

```
VAR wztemporary roll;
...
PROC
  VAR shapedata volume;
  CONST pos t_center := [1000, 1000, 1000];
  ...
  WZCylDef \Inside, volume, t_center, 400, 1000;
  WZLimSup \Temp, roll, volume;
ENDPROC
```

Die `wztemporary`-Variable `roll` wird als Zylinder definiert. Wenn der Roboter dieses Volumen erreicht, wird die Bewegung gestoppt.

Weitere Beispiele

Ein vollständiges Beispiel finden Sie unter der Instruktion `WZDOSet`.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Einschränkungen

`wztemporary`-Daten können als Variable (VAR) oder Persistente (PERS) definiert werden. Sie sind global in einer Task oder lokal in einem Modul möglich, jedoch nicht lokal in einer Routine.

Argumente des Typs `wztemporary` dürfen nur vollständige Daten darstellen, nicht ein Datenfeldelement oder eine Datensatzkomponente.

Eine temporäre Weltzone darf nur in der Bewegungstask definiert (`WZLimSup` oder `WZDOSet`) und gelöscht (`WZFree`) werden. Definitionen von temporären Weltzonen im Hintergrund sind nicht zulässig, da dies andernfalls die Programmabarbeitung in der verbundenen Bewegungstask beeinflusst. Die Instruktionen `WZDisable` und `WZEnable` können in der Hintergrundtask verwendet werden. Wenn eine persistente Variable in einem Multitasking-System verwendet werden muss, setzen Sie den Anfangswert in beiden Tasks auf 0, z. B. PERS wzstationary share_workarea := [0];

Struktur

```
< dataobject of wztemporary >
  < wz of num >
```

Weitere Informationen

Information über	Siehe
World Zones	*Technisches Referenzhandbuch - RAPID Overview*, Abschnitt *Motion and I/O principles - World zones*
Form von Weltzonen	*shapedata – Weltzonen-Formdaten auf Seite 1787*
Stationäre Weltzone	*wzstationary - Stationäre Weltzonendaten auf Seite 1856*
Aktivieren der Überwachung der Weltzonengrenze	*WZLimSup - Aktivieren der Überwachung der Weltzonengrenze auf Seite 1161*
Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs	*WZDOSet - Aktivieren einer Weltzone zum Setzen eines digitalen Ausgangs auf Seite 1144*
Deaktivieren einer Weltzone	*WZDisable - Deaktivieren der temporären Weltzonenüberwachung auf Seite 1142*
Aktivieren einer Weltzone	*WZEnable - Aktivieren einer temporären Weltzonenüberwachung auf Seite 1149*
Löschen einer Weltzone	*WZFree - Löschen der temporären Weltzonenüberwachung auf Seite 1151*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.102 zonedata - Zonendaten

Verwendung

`zonedata` dient der Angabe, wie eine Position erreicht wird, d. h., wie nahe sich die Achsen an der programmierten Position befinden müssen, bevor sie sich zur nächsten Position bewegen.

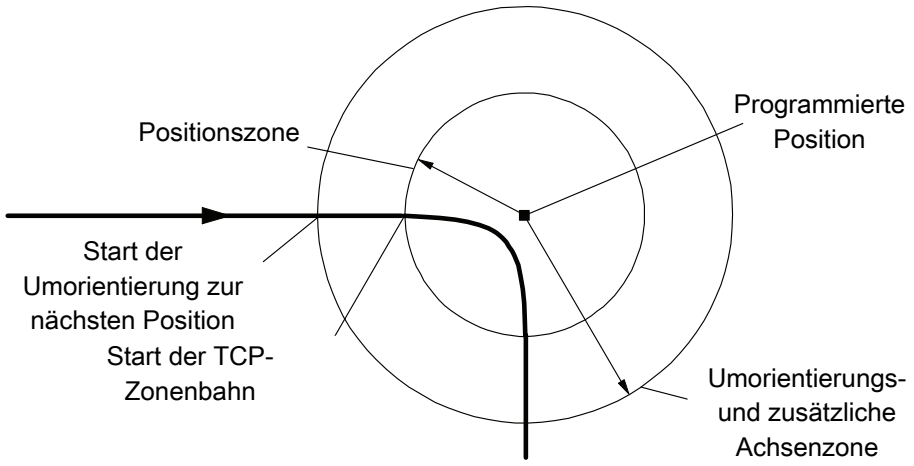
Beschreibung

Eine Position kann als Stopppunkt oder Fly-By-Punkt festgelegt werden.

Ein Stopppunkt bedeutet, dass der Roboter und zusätzliche Achsen die angegebene Position erreichen müssen (Stillstand), bevor die Programmabarbeitung mit der nächsten Instruktion fortfährt. Stopppunkte können auch anders als mit dem vordefinierten `fine` festgelegt werden. Das Stoppkriterium, das mitteilt, ob der Roboter den Punkt erreicht hat, kann über `stoppointdata` manipuliert werden.

Ein Fly-By-Punkt bedeutet, dass die programmierte Position nie erreicht wird. Stattdessen wird vor Erreichen der Position die Bewegungsrichtung geändert. Für jede Position können zwei verschiedene Zonen (Bereiche) definiert werden.

- Positionszone für die TCP-Bahn.
- Umorientierungs- und zusätzliche Achsenzone.



xx1800000945

Zone für die TCP-Bahn

Eine Zonenbahn wird erzeugt, sobald der Zonenrand erreicht ist (siehe obige Abbildung).

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Berechnung der Umorientierungs- und zusätzlichen Achsenzone

Der Datentyp `zonedata` enthält eine Komponente zur Bestimmung der Positionszone, `pzone_tcp`. Die Umorientierungs- und zusätzliche Achsenzone kann durch sämtliche der folgenden `zonedata`-Komponenten beeinflusst werden.

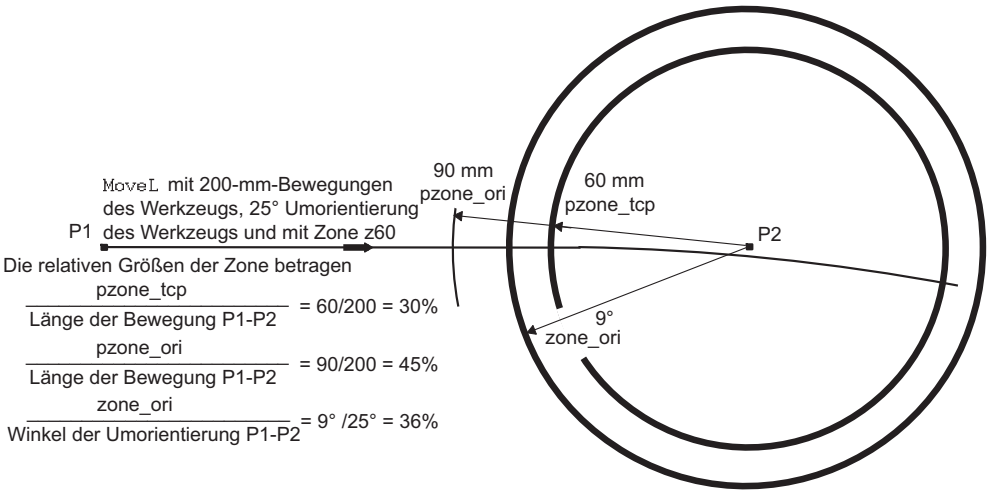
- `pzone_ori` - Zonenradius, in mm, TCP-Bewegung, zur Umorientierung des Werkzeugs
- `pzone_eax` - Zonenradius, in mm, TCP-Bewegung, zur Bewegung der zusätzlichen Achse
- `zone_ori` - Winkelzone, in Grad der Umorientierung des Werkzeugs
- `zone_leax` - Zonengröße, in mm der linearen Bewegung der zusätzlichen Achse
- `zone_reax` - Winkelzone, in Grad der Umorientierung der zusätzlichen Achse durch Drehbewegungen

Die Größe der Umorientierungs- und zusätzlichen Achsenzone wird im Allgemeinen durch die kleinste Zone begrenzt, die aus den oben genannten Komponenten erzeugt wird. Die Zone wird definiert als die kleinste relative Größe der Zone basierend auf den Zonenkomponenten und der programmierten Bewegung.

Wenn die Berechnungen zu einer Umorientierungs- und zusätzlichen Achsenzone führen, die kleiner als der Positionsbereich ist, werden die Umorientierungs- und zusätzliche Achsenzone auf die gleiche Größe wie die Positionszone gesetzt. Die Ausnahme ist, wenn es keine (oder fast keine) TCP-Positionsbewegung gibt. Wenn die Umorientierung groß und die Positionsbewegung klein ist, kann die Positionszone auf die Größe der Umorientierungs- und zusätzlichen Achsenzone reduziert werden.

Umorientierungs- und zusätzliche Achsenzone begrenzt durch `zone_ori`

Die folgende Abbildung zeigt ein Beispiel für eine Umorientierungs- und zusätzliche Achsenzone, die aufgrund von `zone_ori` auf 36 % der Bewegung reduziert wird.



xx0500002362

Vergrößerung der Umorientierungs- und zusätzlichen Achsenzone auf Positionszone

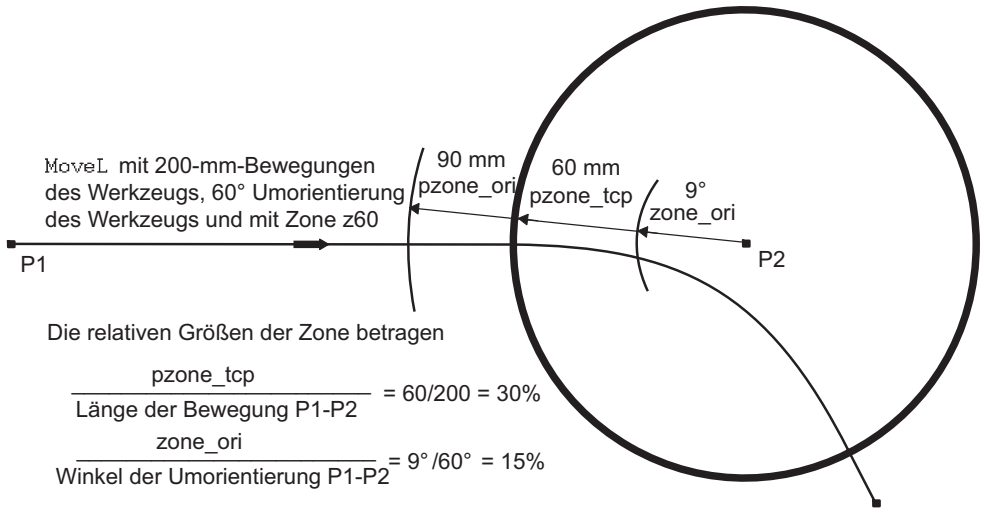
Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.102 zonedata - Zonendaten
RobotWare - OS
Fortsetzung

Die Größe der Umorientierungs- und zusätzliche Achsenzone darf niemals kleiner als die Positionszone sein. Wenn z. B. `zone_ori` eine geringere Größe als `pzone_tcp` ergibt, wird die Umorientierungs- und zusätzliche Achsenzone auf die Größe der Positionszone vergrößert.

Die folgende Abbildung zeigt ein Beispiel, in dem `zone_ori` zu einer Umorientierung- und zusätzlichen Achsenzone mit 15 % der Bewegung führen würde, die aber auf 30 % der Bewegung vergrößert wird, um der Positionszone zu entsprechen.



xx0500002363

Formeln zur Berechnung der Umorientierungs- und zusätzlichen Achsenzone

Im Normalfall können nicht alle `zonedata`-Komponenten angewendet werden. So ist beispielsweise für eine Umorientierung der zusätzlichen Achse durch Drehbewegungen ohne Bewegung des Roboters nur `zone_reax` anwendbar.

Für alle anwendbaren `zonedata`-Komponenten bestimmt die kleinste der folgenden Beziehungen die Größe der Umorientierungs- oder zusätzlichen Achsenzone (sofern sie größer oder gleich der Positionszone ist).

$$\frac{\texttt{pzone\_ori}}{\text{Länge der Bewegung P1–P2}}$$

$$\frac{\texttt{zone\_ori}}{\text{Winkel der Werkzeugumorientierung P1–P2}}$$

$$\frac{\texttt{pzone\_eax}}{\text{Länge der Bewegung P1–P2}}$$

$$\frac{\texttt{pzone\_leax}}{\text{Länge der maximalen linearen zusätzlichen Achsenbewegung P1–P2}}$$

$$\frac{\texttt{pzone\_reax}}{\text{Winkel der max. Umorientierung der zusätzlichen Drehachse P1–P2}}$$

xx1800000781

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Reduzierte Verschleifzonen

Wenn programmierte Positionen nahe beieinander liegen und die Verschleifzonen groß sind, können die Verschleifzonen von der programmierten Größe reduziert werden. Siehe Abschnitt *Interpolation of corner paths* in *Technisches Referenzhandbuch - RAPID Overview*.

Grundlegende Beispiele

Das folgende Beispiel veranschaulicht den Datentyp `zonedata`:

Beispiel 1:

```
VAR zonedata path := [ FALSE, 25, 40, 40, 10, 35, 5 ];
```

Die `path`-Zonendaten werden mit den folgenden Eigenschaften definiert:

- Die Zonengröße für die TCP-Bahn beträgt `25` mm.
- Die Zonengröße für die Werkzeugumorientierung beträgt `40` mm (TCP-Bewegung).
- Die Zonengröße für die externen Achsen beträgt `40` mm (TCP-Bewegung).

Bei einem TCP-Stillstand, einer umfangreichen Umorientierung oder einer großen externen Achsenbewegung gilt hinsichtlich der Zone stattdessen Folgendes:

- Die Zonengröße für die Umorientierung des Werkzeugs ist `10` Grad.
- Die Zonengröße für lineare externe Achsen beträgt `35` mm.
- Die Zonengröße für rotierende externe Achsen beträgt `5` Grad.

Vordefinierte Daten

Einige Zonendaten sind bereits im System definiert.

Stopppunkte

Verwendung von `zonedata` mit Namen `fine`.

Fly-By-Punkte

Bahnzone				Zone		
Name	TCP-Bahn	Ausrichtung	Ext. Achse	Ausrichtung	Linearachse	Rotationsachse
z0	0,3 mm	0,3 mm	0,3 mm	0,03°	0,3 mm	0,03°
z1	1 mm	1 mm	1 mm	0,1°	1 mm	0,1°
z5	5 mm	8 mm	8 mm	0,8°	8 mm	0,8°
z10	10 mm	15 mm	15 mm	1,5°	15 mm	1,5°
z15	15 mm	23 mm	23 mm	2,3°	23 mm	2,3°
z20	20 mm	30 mm	30 mm	3,0°	30 mm	3,0°
z30	30 mm	45 mm	45 mm	4,5°	45 mm	4,5°
z40	40 mm	60 mm	60 mm	6,0°	60 mm	6,0°
z50	50 mm	75 mm	75 mm	7,5°	75 mm	7,5°
z60	60 mm	90 mm	90 mm	9,0°	90 mm	9,0°
z80	80 mm	120 mm	120 mm	12°	120 mm	12°
z100	100 mm	150 mm	150 mm	15°	150 mm	15°

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

3.102 zonedata - Zonendaten
RobotWare - OS
Fortsetzung

Bahnzone				Zone		
Name	**TCP-Bahn**	**Ausrichtung**	**Ext. Achse**	**Ausrichtung**	**Linearachse**	**Rotationsachse**
z150	150 mm	225 mm	225 mm	23°	225 mm	23°
z200	200 mm	300 mm	300 mm	30°	300 mm	30°

Komponenten

`finep`

fine point

Datentyp: `bool`

Definiert, ob die Bewegung mit einem Stopppunkt (`fine` point) oder mit einem Fly-By-Punkt endet.

- `TRUE`: Die Bewegung endet mit einem Stopppunkt und die Programmabarbeitung fährt erst fort, wenn der Roboter den Stopppunkt erreicht. Die übrigen Komponenten in den Zonendaten werden nicht verwendet.
- `FALSE`: Die Bewegung endet als Fly-by-Punkt, und die Programmausführung wird fortgesetzt, wenn die Prefetch-Bedingungen erfüllt sind (siehe Systemparameter *Prefetch Time*).

`pzone_tcp`

path zone TCP

Datentyp: `num`

Die Größe (der Radius) der TCP-Zone in mm.

`pzone_ori`

path zone orientation

Datentyp: `num`

Die Zonengröße (der Radius) zur Umorientierung des Werkzeugs. Die Größe wird als Abstand des TCP vom programmierten Punkt in mm definiert.

Die Größe muss den entsprechenden Wert für `pzone_tcp` überschreiten. Wenn ein niedrigerer Wert angegeben wird, wird die Größe automatisch auf den Wert von `pzone_tcp` erhöht.

`pzone_eax`

path zone external axes

Datentyp: `num`

Die Zonengröße (der Radius) für externe Achsen. Die Größe wird als Abstand des TCP vom programmierten Punkt in mm definiert.

Die Größe muss den entsprechenden Wert für `pzone_tcp` überschreiten. Wenn ein niedrigerer Wert angegeben wird, wird die Größe automatisch auf den Wert von `pzone_tcp` erhöht.

`zone_ori`

zone orientation

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Datentyp: `num`

Zonengröße für die Umorientierung des Werkzeugs in Grad. Wenn der Roboter das Werkobjekt hält, bedeutet dies einen Rotationswinkel für das Werkobjekt.

`zone_leax`

zone linear external axes

Datentyp: `num`

Die Zonengröße für lineare externe Achsen in mm.

`zone_reax`

zone rotational external axes

Datentyp: `num`

Die Zonengröße für rotierende externe Achsen in Grad.

Struktur

```
<data object of zonedata>
  <finep of bool>
  <pzone_tcp of num>
  <pzone_ori of num>
  <pzone_eax of num>
  <zone_ori of num>
  <zone_leax of num>
  <zone_reax of num>
```

Weitere Informationen

Information über	Siehe
Positionierinstruktionen	*Technisches Referenzhandbuch - RAPID Overview*
Bewegungen/Bahnen im Allgemeinen	*Technisches Referenzhandbuch - RAPID Overview*
Konfiguration von externen Achsen	*Anwendungshandbuch - Zusätzliche Achsen und Standalone-Steuerung*
Andere Stopppunkte	*stoppointdata - Stopppunktdaten auf Seite 1799*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Diese Seite wurde absichtlich leer gelassen

4 Programmierungstyp-Beispiele

4.1 ERROR-Behandlungsroutine mit Bewegungen

Verwendung

Diese Programmierungstyp-Beispiele beschreiben die Verwendung von Bewegungsinstruktionen in einer ERROR-Behandlungsroutine nach einem asynchron ausgelösten Prozess- oder Bewegungsfehler.

Diese Funktion kann nur im Haupttask `T_ROB1` oder, bei einem MultiMove System, im Bewegungstask verwendet werden.

Beschreibung

Die `ERROR` -Behandlungsroutine kann eine neue temporäre Bewegung starten und schließlich die ursprüngliche unterbrochene und gestoppte Bewegung neu starten. Beispielsweise kann sie verwendet werden, um eine Wartungsposition anzufahren oder die Zange zu reinigen, nachdem ein asynchron ausgelöster Prozess- oder Bewegungsfehler aufgetreten ist.

Zum Ausführen dieser Funktionalität müssen in der `ERROR`-Behandlungsroutine die Instruktionen `StorePath - RestoPath` verwendet werden. Für den Neustart der Bewegung und die Fortsetzung der Programmabarbeitung sind mehrere `RAPID`-Instruktionen verfügbar.

Programmierungstyp-Beispiele

Programmierungstyp-Beispiele für die Funktionalität werden unten beschrieben.

Prinzip

```
...
ERROR
  IF ERRNO = ERR_PATH_STOP THEN
    StorePath;
    ! Move away and back to the interrupted position
    ...
    RestoPath;
    StartMoveRetry;
  ENDIF
ENDPROC
```

At execution of `StartMoveRetry` setzt der Roboter seine Bewegung fort, ggf. aktive Prozesse werden neu gestartet und das Programm versucht, die Abarbeitung fortzusetzen. Die Funktion von `StartMoveRetry` entspricht der Funktion von `StartMove` plus `RETRY` in einer einzigen Operation.

Automatischer Neustart der Abarbeitung

```
CONST robtarget service_pos := [...];
VAR robtarget stop_pos;
...
ERROR
  IF ERRNO = AW_WELD_ERR THEN
    ! Current movement on motion base path level
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

4.1 ERROR-Behandlungsroutine mit Bewegungen

Path Recovery

Fortsetzung

```
    ! is already stopped.
    ! New motion path level for new movements in the ERROR handler
    StorePath;
    ! Store current position from motion base path level
    stop_pos := CRobT(\Tool:=tool1, \WObj:=wobj1);
    ! Do the work to fix the problem
    MoveJ service_pos, v50, fine, tool1, \WObj:=wobj1;
    ...
    ! Move back to the position on the motion base path level
    MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
    ! Go back to motion base path level
    RestoPath;
    ! Restart the stopped movements on motion base path level,
    ! restart the process and retry program execution
    StartMoveRetry;
  ENDIF
ENDPROC
```

Dies ist ein Programmierungstyp-Beispiel für die Verwendung der automatischen asynchronen Fehlerbehandlung nach einem Prozessfehler während Roboterbewegungen.

Manueller Neustart der Abarbeitung

```
...
ERROR
  IF ERRNO = PROC_ERR_XXX THEN
    ! Current movement on motion base path level
    ! is already stopped and in stop move state.
    ! This error must be handle manually.
    ! Reset the stop move state on motion base path level.
    StopMoveReset;
  ENDIF
ENDPROC
```

Dies ist ein Programmierungstyp-Beispiel für die Verwendung der manuellen asynchronen Fehlerbehandlung nach einem Prozessfehler während Roboterbewegungen.

Nachdem die obige ERROR-Behandlungsroutine vollständig ausgeführt wurde, wird die Programmabarbeitung angehalten und der Programmzeiger befindet sich am Anfang der Instruktion mit dem Prozessfehler (also am Anfang einer beliebigen verwendeten NOSTEPIN-Routine). Der nächste Programmstart startet das Programm und die Bewegung erneut an der Position, an der der ursprüngliche Prozessfehler aufgetreten ist.

Programmabarbeitung

Abarbeitungsverhalten:

- Am Anfang der Abarbeitung der ERROR-Behandlungsroutine verlässt das Programm die Basisabarbeitungsebene.
- Bei der Abarbeitung von StorePath verlässt das Bewegungssystem die Basisabarbeitungsebene.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Bei der Abarbeitung von `RestoPath` kehrt das Bewegungssystem auf die Basisabarbeitungsebene zurück.
- Bei der Abarbeitung von `StartMoveRetry` kehrt das Programm auf die Basisabarbeitungsebene zurück.

Einschränkungen

Die folgenden `RAPID` Instruktionen müssen in einer `ERROR`-Behandlungsroutine mit Bewegungsinstruktionen verwendet werden, um sie nach einem asynchron ausgelösten Prozess- oder Bahnfehler für die automatische Fehlerbehandlung verwenden zu können:

Instruktion	Beschreibung
`StorePath`	Beginn einer neuen Bewegungsbahnebene
`RestoPath`	Rückkehr zur Bewegungsbahn-Basisebene
`StartMoveRetry`	Neustart der unterbrochenen Bewegungen auf der Bewegungsbahn-Basisebene. Außerdem Neustart des Prozesses und der Programmabarbeitung. Die gleiche Funktionalität wie `StartMove` + `RETRY`.

Die folgende `RAPID`-Instruktion muss in der `ERROR`-Behandlungsroutine verwendet werden, um sie nach einem asynchron ausgelösten Prozess- oder Bahnfehler für die manuelle Fehlerbehandlung verwenden zu können:

Instruktion	Beschreibung
`StopMoveReset`	Beginn einer neuen Bewegungsbahnebene

Weitere Informationen

Information über	Siehe
Beginn einer neuen Bewegungsbahnebene	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Rückkehr zur Bewegungsbahn-Basisebene	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Neustart der unterbrochenen Bewegung, des unterbrochenen Prozesses und Wiederholung der Programmabarbeitung	*StartMoveRetry - Erneutes Starten der Roboterbewegung und der Abarbeitung auf Seite 809*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

4.2 Serviceroutinen mit oder ohne Bewegungen

Verwendung

Diese Programmierungstyp-Beispiele beschreiben die Verwendung von Bewegungsinstruktionen in einer Serviceroutine. Das gleiche Prinzip von `StopMove`, `StartMove` und `StopMoveReset` gilt auch für Serviceroutinen ohne Bewegungen (nur logische Instruktionen).

Serviceroutinen und andere Routinen (Prozeduren) ohne Parameter können manuell gestartet werden und Bewegungen entsprechend diesen Programmierungstyp-Beispielen ausführen.

Diese Funktionalität kann nur in der Haupttask `T_ROB1` oder, bei einem MultiMove-System, in Bewegungstasks im unabhängigen oder teilkoordinierten Modus verwendet werden.

Beschreibung

Die Serviceroutine kann eine neue temporäre Bewegung starten und bei einem späteren Programmstart die ursprüngliche Bewegung neu starten. Sie kann z. B. verwendet werden, um eine Wartungsposition anzufahren oder die Schweißzange zu reinigen.

Für diese Funktionalität müssen in der Serviceroutine die Instruktionen `StorePath - RestoPath` und `StopMoveReset` verwendet werden.

Programmierungstyp-Beispiele

Programmierungstyp-Beispiele für die Funktionalität werden unten beschrieben.

Prinzip

```
PROC xxxx()
  StopMove;
  StorePath;
  ! Move away and back to the interrupted position
  ...
  RestoPath;
  StopMoveReset;
ENDPROC
```

`StopMove` ist erforderlich, um zu verhindern, dass die ursprünglich gestoppte Bewegung bei einer manuellen „Programmstopp-Programmneustart“-Sequenz während der Abarbeitung der Serviceroutine neu gestartet wird.

Stopp auf der Bahn

```
VAR robtarget service_pos := [...];
...
PROC proc_stop_on_path()
  VAR robtarget stop_pos;
  ! Current stopped movements on motion base path level
  ! must not be restarted in the service routine.
  StopMove;
  ! New motion path level for new movements in the service routine.
  StorePath;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=tool1 \WObj:=wobj1);
  ! Do the work
  MoveJ service_pos, v50, fine, tool1 \WObj:=wobj1;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for the interrupted movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

In diesem Programmierungstyp-Beispiel werden die Bewegungen in der Serviceroutine an der Position auf der Bahn gestartet und beendet, an der das Programm angehalten wurde.

Beachten Sie außerdem, dass das verwendete Werkzeug und Werkobjekt zum Zeitpunkt der Programmierung bekannt sind.

Stopp am nächsten Stopppunkt

```
TASK PERS tooldata used_tool := [...];
TASK PERS wobjdata used_wobj := [...];
...
PROC proc_stop_in_stop_point()
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level continue to
       its ToPoint and will be finished in a stop point.
  StartMove;
  ! New motion path level for new movements in the service routine
  StorePath;
  ! Get current tool and work object data
  GetSysData used_tool;
  GetSysData used_wobj;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=used_tool \WObj:=used_wobj);
  ! Do the work
  MoveJ Offs(stop_pos,0,0,20),v50,fine,used_tool\WObj:=used_wobj;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, used_tool,\WObj:=used_wobj;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for any new movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

In diesem Programmierungstyp-Beispiel werden die Bewegungen in der Serviceroutine bis zum `ToPoint` in den unterbrochenen Bewegungsinstruktionen fortgesetzt, bevor die Instruktion `StorePath` abgeschlossen wurde.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beachten Sie außerdem, dass das verwendete Werkzeug und Werkobjekt zum Zeitpunkt der Programmierung nicht bekannt sind.

Programmabarbeitung

Abarbeitungsverhalten:

- Am Anfang der Abarbeitung der Serviceroutine verlässt das Programm die Basisabarbeitungsebene.
- Bei der Abarbeitung von `StorePath` verlässt das Bewegungssystem die Basisabarbeitungsebene.
- Bei der Abarbeitung von `RestoPath` kehrt das Bewegungssystem auf die Basisabarbeitungsebene zurück.
- Bei der Abarbeitung von `ENDPROC` kehrt das Programm auf die Basisabarbeitungsebene zurück.

Einschränkungen

Die folgenden `RAPID`-Instruktionen müssen in der Serviceroutine mit Bewegungsinstruktionen verwendet werden, um die Serviceroutine auszuführen:

Instruktion	Beschreibung
`StorePath`	Beginn einer neuen Bewegungsbahnebene
`RestoPath`	Rückkehr zur Bewegungsbahn-Basisebene
`StopMoveReset`	Zurücksetzen des Bewegungsstopp-Status der unterbrochenen Bewegung auf der Bewegungsbahn-Basisebene.

Weitere Informationen

Information über	Siehe
Kein Neustart der bereits gestoppten Bewegung auf der Bewegungsbahn-Basisebene	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Neustart der bereits gestoppten Bewegung auf der Bewegungsbahn-Basisebene	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Beginn einer neuen Bewegungsbahnebene	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Rückkehr zur Bewegungsbahn-Basisebene	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Zurücksetzen des Bewegungsstopp-Status der unterbrochenen Bewegung auf der Bewegungsbahn-Basisebene.	*StopMoveReset - Zurücksetzen des Systemstatus für das Stoppen der Bewegung auf Seite 843*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

4.3 System-E/A-Interrupts mit oder ohne Bewegungen

Verwendung

Diese Programmierungstyp-Beispiele beschreiben die Verwendung von Bewegungsinstruktionen in einer System-E/A-Interrupt-Routine. Das Prinzip von `StopMove`, `StartMove` und `StopMoveReset` gilt auch für System-E/A-Interrupts ohne Bewegungen (nur logische Instruktionen).

Diese Funktionalität kann nur in der Haupttask `T_ROB1` oder, bei einem MultiMove-System, in Bewegungstasks im unabhängigen oder teilkoordinierten Modus verwendet werden.

Beschreibung

Die System-E/A-Interrupt-Routine kann eine neue temporäre Bewegung starten und bei einem späteren Programmstart die ursprüngliche Bewegung neu starten. Die Routine kann z. B. verwendet werden, um eine Wartungsposition anzufahren oder eine Schweißzange zu reinigen, wenn ein Interrupt auftritt.

Für diese Funktionalität müssen in der System-E/A-Interrupt-Routine die Instruktionen `StorePath - RestoPath` und `StopMoveReset` verwendet werden.

Programmierungstyp-Beispiele

Programmierungstyp-Beispiele für die Funktionalität werden unten beschrieben.

Prinzip

```
PROC xxxx()
  StopMove;
  StorePath;
  ! Move away and back to the interrupted position
  ...
  RestoPath;
  StopMoveReset;
ENDPROC
```

`StopMove` ist erforderlich, um zu verhindern, dass die ursprünglich gestoppte Bewegung beim Start der E/A-Interrupt-Routine neu gestartet wird.

Ohne `StopMove` oder mit `StartMove` wird die Bewegung in der E/A-Interrupt-Routine sofort bis zum `ToPoint` in der unterbrochenen Bewegungsinstruktion fortgesetzt.

Stopp auf der Bahn

```
VAR robtarget service_pos := [...];
...
PROC proc_stop_on_path()
  VAR robtarget stop_pos;
  ! Current stopped movements on motion base path level is not
       restarted in the system I/O routine.
  StopMove;
  ! New motion path level for new movements in the system I/O
       routine.
  StorePath;
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

4.3 System-E/A-Interrupts mit oder ohne Bewegungen

Path recovery

Fortsetzung

```
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=tool1 \WObj:=wobj1);
  ! Do the work
  MoveJ service_pos, v50, fine, tool1 \WObj:=wobj1;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for the interrupted movement on motion
       base path level
  StopMoveReset;
ENDPROC
```

In diesem Programmierungstyp-Beispiel werden die unterbrochenen Bewegungen sofort gestoppt und bei einem Programmstart neu gestartet, nachdem die System-E/A-Interrupt-Routine abgeschlossen wurde.

Beachten Sie außerdem, dass das verwendete Werkzeug und Werkobjekt zum Zeitpunkt der Programmierung bekannt sind.

Stopp am nächsten Stopppunkt

```
TASK PERS tooldata used_tool := [...];
TASK PERS wobjdata used_wobj := [...];
...
PROC proc_stop_in_stop_point()
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level continue to
       its ToPoint and will be finished in a stop point.
  StartMove;
  ! New motion path level for new movements in the system
  ! I/O routine
  StorePath;
  ! Get current tool and work object data
  GetSysData used_tool;
  GetSysData used_wobj;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=used_tool \WObj:=used_wobj);
  ! Do the work
  MoveJ Offs(stop_pos,0,0,20),v50,fine,used_tool\WObj:=used_wobj;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, used_tool,\WObj:=used_wobj;
  ! Go back to motion base path level
  RestoPath;
  ! Reset the stop move state for new movement
  ! on motion base path level
  StopMoveReset;
ENDPROC
```

In diesem Programmierungstyp-Beispiel werden die Bewegungen in der System-E/A-Routine bis zum `ToPoint` in den unterbrochenen Bewegungsinstruktionen fortgesetzt.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Beachten Sie außerdem, dass das verwendete Werkzeug und Werkobjekt zum Zeitpunkt der Programmierung nicht bekannt sind.

Programmabarbeitung

Abarbeitungsverhalten:

- Am Anfang der Abarbeitung der System-E/A-Routine verlässt das Programm die Basisabarbeitungsebene.
- Bei der Abarbeitung von `StorePath` verlässt das Bewegungssystem die Basisabarbeitungsebene.
- Bei der Abarbeitung von `RestoPath` kehrt das Bewegungssystem auf die Basisabarbeitungsebene zurück.
- Bei der Abarbeitung von `ENDPROC` kehrt das Programm auf die Basisabarbeitungsebene zurück.

Einschränkungen

Die folgenden `RAPID`-Instruktionen müssen in der System-E/A-Routine mit Bewegungsinstruktionen verwendet werden, um die System-E/A-Routine auszuführen:

Instruktion	Beschreibung
`StorePath`	Beginn einer neuen Bewegungsbahnebene
`RestoPath`	Rückkehr zur Bewegungsbahn-Basisebene
`StopMoveReset`	Zurücksetzen des Bewegungsstopp-Status der unterbrochenen Bewegung auf der Bewegungsbahn-Basisebene.

Weitere Informationen

Information über	Siehe
Kein Neustart der bereits gestoppten Bewegung auf der Bewegungsbahn-Basisebene	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Neustart der bereits gestoppten Bewegung auf der Bewegungsbahn-Basisebene	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806*
Beginn einer neuen Bewegungsbahnebene	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Rückkehr zur Bewegungsbahn-Basisebene	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Zurücksetzen des Bewegungsstopp-Status der unterbrochenen Bewegung auf der Bewegungsbahn-Basisebene.	*StopMoveReset - Zurücksetzen des Systemstatus für das Stoppen der Bewegung auf Seite 843*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

4.4 TRAP-Routinen mit Bewegungen

Verwendung

Diese Programmierungstyp-Beispiele beschreiben die Verwendung von Bewegungsinstruktionen in einer TRAP-Routine nach einem Interrupt.

Diese Funktionalität kann nur in der Haupttask T_ROB1 oder, bei einem MultiMove-System, in Bewegungstasks verwendet werden.

Beschreibung

Die TRAP-Routine kann eine neue temporäre Bewegung starten und schließlich die ursprüngliche Bewegung neu starten. Die Routine kann z. B. verwendet werden, um eine Wartungsposition anzufahren oder eine Schweißzange zu reinigen, wenn ein Interrupt auftritt.

Zum Ausführen dieser Funktionalität müssen in der TRAP-Routine die Instruktionen StorePath - RestoPath und StartMove verwendet werden.

Programmierungstyp-Beispiele

Programmierungstyp-Beispiele für die Funktionalität werden unten beschrieben.

Prinzip

```
TRAP xxxx
  StopMove;
  StorePath;
  ! Move away and back to the interrupted position
  ...
  RestoPath;
  StartMove;
ENDTRAP
```

Bei Verwendung von StopMove wird die Bewegung sofort auf der Bahn angehalten. Anderenfalls wird die Bewegung bis zum ToPoint in der aktuellen Bewegungsinstruktion fortgesetzt.

Stopp am nächsten Stopppunkt

```
VAR robtarget service_pos := [...];
...
TRAP trap_in_stop_point
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level continue
  ! to it's ToPoint and will be finished in a stop point.
  ! New motion path level for new movements in the TRAP
  StorePath;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=tool1 \WObj:=wobj1);
  ! Do the work
  MoveJ service_pos, v50, fine, tool1 \WObj:=wobj1;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, tool1, \WObj:=wobj1;
  ! Go back to motion base path level
```

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

```
  RestoPath;
  ! Restart the interupted movements on motion base path level
  StartMove;
ENDTRAP
```

In diesem Programmierungstyp-Beispiel starten und enden die Bewegungen in der TRAP-Routine am `ToPoint` in den unterbrochenen Bewegungsinstruktionen. Beachten Sie außerdem, dass das Werkzeug und Werkobjekt zum Zeitpunkt der Programmierung bekannt sind.

Sofortiger Stopp auf der Bahn

```
TASK PERS tooldata used_tool := [...];
TASK PERS wobjdata used_wobj := [...];
...
TRAP trap_stop_at_once
  VAR robtarget stop_pos;
  ! Current move instruction on motion base path level stops
  ! at once
  StopMove;
  ! New motion path level for new movements in the TRAP
  StorePath;
  ! Get current tool and work object data
  GetSysData used_tool;
  GetSysData used_wobj;
  ! Store current position from motion base path level
  stop_pos := CRobT(\Tool:=used_tool \WObj:=used_wobj);
  ! Do the work
  MoveJ Offs(stop_pos,0,0,20),v50,fine,used_tool\WObj:=used_wobj;
  ...
  ! Move back to interrupted position on the motion base path level
  MoveJ stop_pos, v50, fine, used_tool,\WObj:=used_wobj;
  ! Go back to motion base path level
  RestoPath;
  ! Restart the interupted movements on motion base path level
  StartMove;
ENDTRAP
```

In diesem Programmierungstyp-Beispiel werden die Bewegungen in der TRAP-Routine an der Position auf der Bahn gestartet und beendet, an der die unterbrochene Bewegungsinstruktion angehalten wurde. Beachten Sie außerdem, dass das verwendete Werkzeug und Werkobjekt zum Zeitpunkt der Programmierung nicht bekannt sind.

Programmabarbeitung

Abarbeitungsverhalten:

- Am Anfang der Abarbeitung der `TRAP`-Routine verlässt das Programm die Basisabarbeitungsebene.
- Bei der Abarbeitung von `StorePath` verlässt das Bewegungssystem die Basisabarbeitungsebene.
- Bei der Abarbeitung von `RestoPath` kehrt das Bewegungssystem auf die Basisabarbeitungsebene zurück.

Fortsetzung auf nächster Seite

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

- Bei der Abarbeitung von ENDTRAP kehrt das Programm auf die Basisabarbeitungsebene zurück.

Einschränkungen

Die folgenden RAPID-Instruktionen müssen in der TRAP-Routine mit Bewegungsinstruktionen verwendet werden, um die Routine auszuführen:

Instruktion	Beschreibung
StorePath	Beginn einer neuen Bewegungsbahnebene
RestoPath	Rückkehr zur Bewegungsbahn-Basisebene
StartMove	Neustart der unterbrochenen Bewegungen auf der Bewegungsbahn-Basisebene

Weitere Informationen

Information über	Siehe
Sofortiges Anhalten der aktuellen Bewegung	*StopMove - Stoppen der Roboterbewegung auf Seite 839*
Beginn einer neuen Bewegungsbahnebene	*StorePath - Speichern der Bahn bei einem Interrupt auf Seite 846*
Rückkehr zur Bewegungsbahn-Basisebene	*RestoPath - Wiederherstellen der Bahn nach einem Interrupt auf Seite 620*
Neustart der unterbrochenen Bewegung	*StartMove - Erneutes Starten der Roboterbewegung auf Seite 806*

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Index

(ConfL, 158
(MoveCDO, 403

A
Abs, 1167
AbsDnum, 1169
AccSet, 25
ACos, 1171
ACosDnum, 1172
ActEventBuffer, 28
ActUnit, 30
Add, 32
AInput, 1173
aiotrigg, 1635
ALIAS, 1637
AliasCamera, 34
AliasIO, 37
AliasIOReset, 40
AND, 1175
AOutput, 1177
ArgName, 1179
ASin, 1183
ASinDnum, 1184
Assignment
=, 42
ATan, 1185
ATan2, 1187
ATan2Dnum, 1188
ATanDnum, 1186

B
BitAnd, 1189
BitAndDnum, 1191
BitCheck, 1193
BitCheckDnum, 1195
BitClear, 44
BitLSh, 1197
BitLShDnum, 1199
BitNeg, 1202
BitNegDnum, 1204
BitOr, 1206
BitOrDnum, 1208
BitRSh, 1210
BitRShDnum, 1212
BitSet, 47
BitXOr, 1214
BitXOrDnum, 1216
BookErrNo, 50
bool, 1638
Break, 52
btnres, 1639
busstate, 1641
buttondata, 1642
byte, 1644
ByteToStr, 1218

C
CalcJointT, 1220
CalcRobT, 1225
CalcRotAxFrameZ, 1228
CalcRotAxisFrame, 1233
CallByVar, 53
cameradev, 1645
camerastatus, 1646
cameratarget, 1648
CamFlush, 55
CamGetExposure, 1237
CamGetLoadedJob, 1239
CamGetMode, 1241
CamGetName, 1242
CamGetParameter, 56
CamGetResult, 58
CamLoadJob, 60
CamNumberOfResults, 1243
CamReqImage, 62
CamSetExposure, 64
CamSetParameter, 66
CamSetProgramMode, 68
CamSetRunMode, 70
CamStartLoadJob, 72
CamStartSetParameter, 74
CamWaitLoadJob, 77
CamWaitSetParameter, 79
CancelLoad, 81
capaptrreferencedata, 1651
CapAPTrSetup, 83
CapAPTrSetupAI, 86
CapAPTrSetupAO, 89
CapAPTrSetupPERS, 92
CapC, 95
CapCondSetDO, 105
capdata, 1653
CapEquiDist, 107
CapGetFailSigs, 1245
CapL, 109
caplatrackdata, 1657
CapLATrSetup, 118
CapNoProcess, 123
CapRefresh, 125
CAPSetStopMode, 127
capspeeddata, 1661
capstopmode, 1663
captrackdata, 1664
capweavedata, 1667
CapWeaveSync, 128
CASE, 879
CDate, 1247
cfgdomain, 1675
CheckProgRef, 131
CirPathMode, 133
CJointT, 1248
Clear, 139
ClearIOBuff, 140
ClearPath, 142
ClearRawBytes, 146
ClkRead, 1250
ClkReset, 148
ClkStart, 149
ClkStop, 151
clock, 1676
Close, 152
CloseDir, 153
comment, 154
CompactIF, 155
confdata, 1677
ConfJ, 156
CONNECT, 161
ContactL, 163
CopyFile, 169
CopyRawBytes, 171

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

CornerPathWarning, 174
CorrClear, 176
CorrCon, 177
corrdescr, 1685
CorrDiscon, 182
CorrRead, 1252
CorrWrite, 183
Cos, 1253
CosDnum, 1254
CPos, 1255
CRobT, 1257
CrossProd, 1261
CSpeedOverride, 1264
CTime, 1266
CTool, 1267
CWObj, 1269

D
datapos, 1687
DeactEventBuffer, 185
DeactUnit, 187
Decr, 189
DecToHex, 1271
DefAccFrame, 1272
DefDFrame, 1275
DefFrame, 1278
Dim, 1282
DInput, 1284
dionum, 1688
dir, 1689
Distance, 1286
DIV, 1288
dnum, 1690
DnumToNum, 1289
DnumToStr, 1291
DotProd, 1293
DOutput, 1295
DropSensor, 191
DropWObj, 193

E
E/A-Interrupt-Routinen, 1873
ELSE, 258
ELSEIF, 258
ENDIF, 258
EOF_NUM, 1464
EOffsOff, 194
EOffsOn, 195
EOffsSet, 198
EraseModule, 200
errdomain, 1692
ErrLog, 202
errnum, 1694
ERROR-Behandlungsroutine, 1867
ErrRaise, 206
errstr, 1702
errtype, 1703
ErrWrite, 211
EulerZYX, 1297
event_type, 1704
EventType, 1299
exec_level, 1705
ExecHandler, 1301
ExecLevel, 1302
EXIT, 213
ExitCycle, 214
Exp, 1303
extjoint, 1706

F
FileSize, 1304
FileTimeDnum, 1307
FitCircle, 216
flypointdata, 1708
FOR, 220
FricIdEvaluate, 223
FricIdInit, 222
FricIdSetFricLevels, 226
FSSize, 1310

G
GetAxisDistance, 1313
GetAxisMoveTime, 1315
GetDataVal, 228
GetGroupSignalInfo, 231
GetJointData, 233
GetMaxNumberOfCyclicBool, 1317
GetMecUnitName, 1318
GetModalPayLoadMode, 1319
GetMotorTorque, 1320
GetNextCyclicBool, 1323
GetNextMechUnit, 1325
GetNextSym, 1328
GetNumberOfCyclicBool, 1330
GetServiceInfo, 1331
GetSignalOrigin, 1333
GetSysData, 235
GetSysInfo, 1335
GetTaskName, 1338
GetTime, 1340
GetTorqueMargin, 1342
GetTrapData, 238
GetTSPStatus, 1344
GetUASUserName, 1346
GInput, 1348
GInputDnum, 1350
GOTO, 240
GOutput, 1353
GOutputDnum, 1355
GripLoad, 242

H
handler_type, 1711
HexToDec, 1358
HollowWristReset, 244

I
ICap, 246
icondata, 1712
IDelete, 252
identno, 1714
IDisable, 253
IEnable, 254
IError, 255
IF, 258
Incr, 260
IndAMove, 262
IndCMove, 266
IndDMove, 270
IndInpos, 1359
IndReset, 274
IndRMove, 279
IndSpeed, 1361
InitSuperv, 284

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

Interrupt-Routinen, 1876
intnum, 1716
InvertDO, 285
IOBusStart, 287
IOBusState, 288
iodev, 1718
IODisable, 291
IOEnable, 294
IOEventMessage, 297
iounit_state, 1719
IOUnitState, 1363
IPathPos, 299
IPers, 301
IRMQMessage, 303
IsBrakeCheckActive, 1366
IsCollFree, 1367
IsCyclicBool, 1369
IsFile, 1372
ISignalAI, 307
ISignalAO, 316
ISignalDI, 320
ISignalDO, 324
ISignalGI, 328
ISignalGO, 331
IsLeadThrough, 1376
ISleep, 334
IsMechUnitActive, 1378
IsPers, 1379
IsStopMoveAct, 1381
IsStopStateEvent, 1383
IsSyncMoveOn, 1385
IsSysId, 1387
IsVar, 1388
ITimer, 336
IVarValue, 339
IWatch, 343

J
jointtarget, 1720

L
label, 345
listitem, 1722
Load, 346
loaddata, 1723
LoadId, 351
loadidnum, 1729
loadsession, 1730

M
MakeDir, 357
ManLoadIdProc, 359
MatrixSolve, 363
MatrixSolveQR, 366
MatrixSVD, 368
Max, 1390
MaxExtLinearSpeed , 1391
MaxExtReorientSpeed, 1392
MaxRobReorientSpeed, 1393
MaxRobSpeed, 1394
MechUnitLoad, 372
mecunit, 1731
Min, 1395
MirPos, 1396
MOD, 1398
ModExist, 1399
ModTimeDnum, 1400
MotionPlannerNo, 1402
MotionProcessModeSet, 377
MotionSup, 379
motsetdata, 1733
MoveAbsJ, 382
MoveC, 389
MoveCAO, 398
MoveCGO, 408
MoveCSync, 413
MoveExtJ, 418
MoveJ, 422
MoveJAO, 428
MoveJDO, 433
MoveJGO, 438
MoveJSync, 443
MoveL, 448
MoveLAO, 455
MoveLDO, 460
MoveLGO, 465
MoveLSync, 470
MovePnP, 475
MToolRotCalib, 486
MToolTCPCalib, 489

N
NonMotionMode, 1404
NOrient, 1407
NOT, 1406
num, 1740
NumToDnum, 1409
NumToStr, 1410

O
Offs, 1412
opcalc, 1742
Open, 492
OpenDir, 497
OpMode, 1414
opnum, 1743
OR, 1415
orient, 1744
OrientZYX, 1417
ORobT, 1419

P
PackDNHeader, 499
PackRawBytes, 502
paridnum, 1749
ParIdPosVaild, 1421
ParIdRobValid, 1424
paridvalidnum, 1751
PathAccLim, 507
PathLengthGet, 1427
PathLengthReset, 511
PathLengthStart, 513
PathLengthStop, 515
PathLevel, 1429
pathrecid, 1753
PathRecMoveBwd, 517
PathRecMoveFwd, 525
PathRecStart, 528
PathRecStop, 531
PathRecValidBwd, 1431
PathRecValidFwd, 1434
PathResol, 534
PDispOff, 536
PDispOn, 537

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

PDispSet, 542
PFRestart, 1438
pnpdata, 1755
pos, 1757
pose, 1759
PoseInv, 1439
PoseMult, 1441
PoseVect, 1443
Pow, 1445
PowDnum, 1446
PPMovedInManMode, 1447
Present, 1448
ProcCall, 545
ProcerrRecovery, 547
processtimes, 1760
progdisp, 1761
ProgMemFree, 1450
PrxActivAndStoreRecord, 553
PrxActivRecord, 555
PrxDbgStoreRecord, 557
PrxDeactRecord, 558
PrxGetMaxRecordpos, 1451
PrxResetPos, 559
PrxResetRecords, 560
PrxSetPosOffset, 561
PrxSetRecordSampleTime, 562
PrxSetSyncalarm, 563
PrxStartRecord, 565
PrxStopRecord, 567
PrxStoreRecord, 568
PrxUseFileRecord, 570
PulseDO, 572

R
RAISE, 575
RaiseToUser, 578
Rand, 1452
rawbytes, 1764
RawBytesLen, 1454
ReadAnyBin, 581
ReadBin, 1456
ReadBlock, 584
ReadCfgData, 586
ReadDir, 1459
ReadErrData, 590
ReadMotor, 1462
ReadNum, 1464
ReadRawBytes, 593
ReadStr, 1467
ReadStrBin, 1472
ReadVar, 1474
ReadVarArr, 596
RelTool, 1476
RemainingRetries, 1478
RemoveAllCyclicBool, 599
RemoveCyclicBool, 601
RemoveDir, 603
RemoveFile, 605
RemoveSuperv, 607
RenameFile, 609
Reset, 611
ResetAxisDistance, 613
ResetAxisMoveTime, 615
ResetPPMoved, 617
ResetRetryCount, 618
ResetTorqueMargin, 619
restartblkdata, 1766
restartdata, 1768
RestoPath, 620
RETRY, 622
RETURN, 624
Rewind, 626
RMQEmptyQueue, 627
RMQFindSlot, 629
RMQGetMessage, 631
RMQGetMsgData, 634
RMQGetMsgHeader, 637
RMQGetSlotName, 1479
rmqheader, 1772
rmqmessage, 1774
RMQReadWait, 640
RMQSendMessage, 643
RMQSendWait, 647
rmqslot, 1776
robjoint, 1777
RobName, 1481
RobOS, 1483
robtarget, 1778
Round, 1484
RoundDnum, 1486
RunMode, 1488

S
SafetyControllerGetChecksum, 1490
SafetyControllerGetOpModePinCode, 1491
SafetyControllerGetSWVersion, 1492
SafetyControllerGetUserChecksum, 1493
SafetyControllerSyncRequest, 652
Save, 653
SaveCfgData, 656
SCWrite, 658
SearchC, 661
SearchExtJ, 672
SearchL, 681
SenDevice, 693
sensor, 1782
Sensor Interface, 584
sensorstate, 1784
sensorvardata, 1785
Serviceroutinen, 1870
Set, 695
SetAllDataVal, 697
SetAO, 700
SetDataSearch, 702
SetDataVal, 707
SetDO, 710
SetGO, 713
SetLeadThrough, 717
SetSysData, 720
SetupCyclicBool, 722
SetupSuperv, 725
shapedata, 1787
SiClose, 731
SiConnect, 728
SiGetCyclic, 733
signalorigin, 1789
signalxx, 1791
SimCollision, 735
Sin, 1494
SinDnum, 1495
SingArea, 736
SiSetCyclic, 739
SkipWarn, 741

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

SocketAccept, 742
SocketBind, 746
SocketClose, 749
SocketConnect, 751
SocketCreate, 754
socketdev, 1793
SocketGetStatus, 1496
SocketListen, 757
SocketPeek, 1499
SocketReceive, 759
SocketReceiveFrom, 764
SocketSend, 769
SocketSendTo, 774
socketstatus, 1794
SoftAct, 779
SoftDeact, 781
SoftElbow, 782
speeddata, 1795
SpeedLimAxis, 784
SpeedLimCheckPoint, 789
SpeedRefresh, 795
SpyStart, 798
SpyStop, 801
Sqrt, 1502
SqrtDnum, 1503
StartLoad, 802
StartMove, 806
StartMoveRetry, 809
STCalcForce, 1504
STCalcTorque, 1506
STCalib, 812
STClose, 817
StepBwdPath, 821
STIndGun, 24, 823
STIndGunReset, 24, 825
STIsCalib, 1508
STIsClosed, 1510
STIsIndGun, 1512
STIsOpen, 1513
SToolRotCalib, 826
SToolTCPCalib, 829
Stop, 833
STOpen, 836
StopMove, 839
StopMoveReset, 843
stoppointdata, 1799
StorePath, 846
StrDigCalc, 1515
StrDigCmp, 1518
StrFind, 1520
StrFormat, 1522
string, 1806
stringdig, 1808
StrLen, 1524
StrMap, 1525
StrMatch, 1527
StrMemb, 1529
StrOrder, 1531
StrPart, 1533
StrToByte, 1535
StrToVal, 1537
STTune, 849
STTuneReset, 853
supervtimeouts, 1809
SupSyncSensorOff, 854
SupSyncSensorOn, 855
switch, 1811
symnum, 1812
syncident, 1814
SyncMoveOff, 857
SyncMoveOn, 863
SyncMoveResume, 869
SyncMoveSuspend, 871
SyncMoveUndo, 873
SyncToSensor, 875
system data, 1815
SystemStopAction, 877

T

Tan, 1539
TanDnum, 1540
taskid, 1818
TaskIsActive, 1545
TaskIsExecuting, 1547
TaskRunMec, 1541
TaskRunRob, 1542
tasks, 1819
TasksInSync, 1543
TEST, 879
TestAndSet, 1549
TestDI, 1552
testsignal, 1821
TestSignDefine, 881
TestSignRead, 1554
TestSignReset, 883
TextGet, 1556
TextTabFreeToUse, 1559
TextTabGet, 1561
TextTabInstall, 884
tooldata, 1823
TPErase, 886
tpnum, 1830
TPReadDnum, 887
TPReadFK, 891
TPReadNum, 896
TPShow, 900
TPWrite, 901
trapdata, 1831
TriggC, 904
TriggCheckIO, 913
triggdata, 1833
TriggDataCopy, 919
TriggDataReset, 921
TriggDataValid, 1563
TriggEquip, 923
TriggInt, 930
TriggIO, 935
triggios, 1835
triggiosdnum, 1838
TriggJ, 942
TriggL, 950
TriggLIOs, 959, 967
triggmode, 1840
TriggRampAO, 975
TriggSpeed, 983
TriggStopProc, 994
triggstrgo, 1843
Trunc, 1565
TruncDnum, 1567
TryInt, 1000
TRYNEXT, 1002
tsp_status, 1846
TuneReset, 1003

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

TuneServo, 1004
tunetype, 1848
Type, 1569

U
UIAlphaEntry, 1571
UIClientExist, 1578
UIDnumEntry, 1580
UIDnumTune, 1588
UIListView, 1596
UIMessageBox, 1605
UIMsgBox, 1011
UIMsgWrite, 1020
UIMsgWriteAbort, 1025
UINumEntry, 1614
UINumTune, 1621
UIShow, 1026
uishownum, 1849
UnLoad, 1030
UnpackRawBytes, 1033

V
ValidIO, 1628
ValToStr, 1630
VectMagn, 1632
VelSet, 1038

W
WaitAI, 1040
WaitAO, 1046
WaitDI, 1052
WaitDO, 1058
WaitGI, 1064
WaitGO, 1071
WaitLoad, 1078
WaitRob, 1083
WaitSensor, 1085
WaitSyncTask, 1089
WaitTestAndSet, 1093
WaitTime, 1096
WaitUntil, 1098
WaitWObj, 1106
WarmStart, 1109
weavestartdata, 1850
WHILE, 1110
wobjdata, 1852
WorldAccLim, 1112
Write, 1114
WriteAnyBin, 1117
WriteBin, 1119
WriteBlock, 1121
WriteCfgData, 1123
WriteRawBytes, 1127
WriteStrBin, 1130
WriteVar, 1132
WriteVarArr, 1135
WZBoxDef, 1137
WZCylDef, 1139
WZDisable, 1142
WZDOSet, 1144
WZEnable, 1149
WZFree, 1151
WZHomeJointDef, 1153
WZLimJointDef, 1157
WZLimSup, 1161
WZSphDef, 1164
wzstationary, 1856
wztemporary, 1858

X
XOR, 1634

Z
zonedata, 1860

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.

ABB AB
Robotics & Discrete Automation
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

ABB AS
Robotics & Discrete Automation
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.
Robotics & Discrete Automation
No. 4528 Kangxin Highway
PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

ABB Inc.
Robotics & Discrete Automation
1250 Brown Road
Auburn Hills, MI 48326
USA
Telephone: +1 248 391 9000

abb.com/robotics

3HAC050917-003, Rev P, de

© Copyright 2004-2021 ABB. Alle Rechte vorbehalten.
Spezifikationen können ohne vorherige Ankündigung geändert werden.