

REFERENZANLEITUNG VAL3

Version 8

Zusätze und "Errata" finden Sie in dem mit der CD-ROM des Controllers gelieferten Dokument "readme.pdf".

INHALTSVERZEICHNIS

| | |
|--|-----------|
| 1 - EINLEITUNG | 13 |
| 2 - ELEMENTE DER SPRACHE VAL 3 | 17 |
| 2.1 APPLIKATIONEN | 19 |
| 2.1.1 Definition | 19 |
| 2.1.2 Vorprogrammierte Inhalte..... | 19 |
| 2.1.3 Starten und Beenden der Applikation..... | 19 |
| 2.1.4 Parameter der Applikation..... | 19 |
| 2.1.4.1 Längeneinheit..... | 20 |
| 2.1.4.2 Größe des Laufzeitspeichers | 20 |
| 2.1.4.3 Applikation grafische Benutzerschnittstelle (Benutzerseiten) | 20 |
| 2.2 PROGRAMME..... | 21 |
| 2.2.1 Definition | 21 |
| 2.2.2 Wiedereinsprung | 21 |
| 2.2.3 Programm Start()..... | 21 |
| 2.2.4 Programm Stop() | 21 |
| 2.2.5 Anweisungen zur Programmsteuerung | 22 |
| Comment // | 22 |
| call program | 22 |
| return | 22 |
| if control instruction | 23 |
| while control instruction | 24 |
| do ... until control instruction | 24 |
| for control instruction | 25 |
| switch control instruction | 26 |
| 2.3 DATENELEMENTE..... | 28 |
| 2.3.1 Definition | 28 |
| 2.3.2 Einfache Typen | 28 |
| 2.3.3 Strukturierte Typen..... | 28 |
| 2.3.4 Container | 29 |
| 2.4 DATENINITIALISIERUNG | 29 |
| 2.4.1 Daten einfachen Typs | 29 |
| 2.4.2 Daten strukturierten Typs | 29 |
| 2.5 VARIABLEN..... | 30 |
| 2.5.1 Definition | 30 |
| 2.5.2 Gültigkeitsbereich einer Variablen..... | 30 |
| 2.5.3 Zugriff auf den Wert einer Variablen..... | 30 |
| 2.5.4 Für alle Variablentypen geltende Befehle..... | 31 |
| num size(*) | 31 |
| bool isDefined(*) | 31 |
| bool insert(*) | 32 |

| | |
|---|-----------|
| bool delete(*) | 33 |
| num getData (string sDataName, *) | 34 |
| 2.5.5 Für die array-variablen spezifische anweisungen | 35 |
| void append(*) | 35 |
| num size(*, num nDimension) | 35 |
| void resize(*, num nDimension, num nSize) | 35 |
| 2.5.6 Spezifische Anweisungen für Collection-Variablen..... | 36 |
| string first() | 36 |
| string next() | 36 |
| string last() | 36 |
| string prev() | 36 |
| 2.6 PROGRAMMPARAMETER | 37 |
| 2.6.1 Parameter per Elementwert | 38 |
| 2.6.2 Parameter per Elementreferenz | 38 |
| 2.6.3 Parameter per Array- oder Collection-Referenz | 39 |
| 3 - EINFACHE TYPEN..... | 41 |
| 3.1 TYP BOOL | 43 |
| 3.1.1 Definition | 43 |
| 3.1.2 Operatoren | 43 |
| 3.2 TYP NUM | 44 |
| 3.2.1 Definition | 44 |
| 3.2.2 Operatoren | 45 |
| 3.2.3 Anweisungen..... | 46 |
| num sin(num nAngle) | 46 |
| num asin(num nValue) | 46 |
| num cos(num nAngle) | 46 |
| num acos(num nValue) | 46 |
| num tan(num nAngle) | 46 |
| num atan(num nValue) | 47 |
| num abs(num nValue) | 47 |
| num sqrt(num nValue) | 47 |
| num exp(num nValue) | 47 |
| num power(num nX, num nY) | 47 |
| num ln(num nValue) | 48 |
| num log(num nValue) | 48 |
| num roundUp(num nValue) | 48 |
| num roundDown(num nValue) | 48 |
| num round(num nValue) | 49 |
| num min(num nX, num nY) | 49 |
| num max(num nX, num nY) | 49 |
| num limit(num nValue, num nMin, num nMax) | 49 |
| num sel(bool bCondition, num nValue1, num nValue2) | 49 |
| 3.3 BITFIELD-TYP | 50 |
| 3.3.1 Definition | 50 |
| 3.3.2 Operatoren | 50 |
| 3.3.3 Anweisungen..... | 50 |
| num bNot(num nBitField) | 50 |
| num bAnd(num nBitField1, num nBitField2) | 50 |
| num bOr(num nBitField1, num nBitField2) | 51 |
| num bXor(num nBitField1, num nBitField2) | 51 |
| num toBinary(num nValue[], num nValueSize, String sDataFormat, num& nDataByte[]) | 52 |
| num fromBinary(num nDataByte[], num nDataSize, String sDataFormat, num& nValue[]) | 52 |

| | |
|---|-----------|
| 3.4 TYP STRING | 54 |
| 3.4.1 Definition | 54 |
| 3.4.2 Operatoren | 54 |
| 3.4.3 Anweisungen | 54 |
| string toString(string sFormat, num nValue) | 54 |
| string toNum(string sString, num& nValue, bool& bReport) | 55 |
| string chr(num nCodePoint) | 56 |
| num asc(string sText, num nPosition) | 57 |
| string left(string sText, num nSize) | 57 |
| string right(string sText, num nSize) | 57 |
| string mid(string sText, num nSize, num nPosition) | 57 |
| string insert(string sText, string sInsertion, num nPosition) | 58 |
| string delete(string sText, num nSize, num nPosition) | 58 |
| string replace(string sText, string sReplacement, num nSize, num nPosition) | 58 |
| num find(string sText1, string sText2) | 58 |
| num len(string sText) | 58 |
| 3.5 TYP DIO | 59 |
| 3.5.1 Definition | 59 |
| 3.5.2 Operatoren | 59 |
| 3.5.3 Anweisungen | 60 |
| void dioLink(dio& diVariable, dio diSource) | 60 |
| num dioGet(dio diArray[]) | 60 |
| num dioSet(dio diArray[], num nValue) | 61 |
| num ioStatus(dio diInputOutput) | 61 |
| num ioStatus(dio diInputOutput, string& sDescription, string& sPhysicalPath) | 62 |
| 3.6 TYP AIO | 63 |
| 3.6.1 Definition | 63 |
| 3.6.2 Anweisungen | 63 |
| void aioLink(aio& aiVariable, aio aiSource) | 63 |
| num aioGet(aio aiInput) | 63 |
| num aioSet(aio aiOutput, num nValue) | 64 |
| num ioStatus(aio aiInputOutput) | 64 |
| num ioStatus(aio diInputOutput, string& sDescription, string& sPhysicalPath) | 65 |
| 3.7 TYP SIO..... | 66 |
| 3.7.1 Definition | 66 |
| 3.7.1.1 Sockets | 66 |
| 3.7.1.2 Socket-Parameter | 66 |
| 3.7.1.3 TCP (Transmission Control Protocol) | 67 |
| 3.7.1.4 UDP (User Diagram Protocol) | 67 |
| 3.7.2 Operatoren | 68 |
| 3.7.3 Anweisungen | 69 |
| void sioLink(sio& siVariable, sio siSource) | 69 |
| num clearBuffer(sio siVariable) | 69 |
| num ioStatus(sio siInputOutput) | 69 |
| num ioStatus(sio siInputOutput, string& sDescription, string& sPhysicalPath) | 69 |
| num sioGet(sio siInput, num& nData[]) | 70 |
| num sioSet(sio siOutput, num& nData[], ,nNbElements) | 70 |
| num sioCtrl(sio siChannel, string nParameter, *value) | 71 |

4 - BENUTZERSCHNITTSTELLE 73

| | |
|---|-----------|
| 4.1 ANWEISUNGEN | 75 |
| num userPage(string sPageName) | 75 |
| void userPageBind (string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib, num& nVar, num nSize, string sDirection, num nRefresh) | |
| void userPageBind (string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib, string& nVar, num nSize, string sDirection, num nRefresh) | |
| void userPageBind (string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib, bool& nVar, num nSize, string sDirection, num nRefresh) | |
| void userPageBind (string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib, dio& dDio, num nSize, string sDirection, num nRefresh) | |
| void userPageBind (string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib, aio& aAio, num nSize, string sDirection, num nRefresh) | 76 |
| bool userPageUnbind(string sPage, string sGraphicalObjectId, string sGraphicalObjectAttrib) | 77 |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, bool& bVar, bool& bValue) | |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, dio& bVar, bool& bValue) | |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, num& nVar, num& nValue) | |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, aio& aVar, num& bValue) | |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, string& sVar, string& sValue) | |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, bool& bVar) | |
| void userPageBindXEVENTX(string sPage, string sGraphicalObjectId, dio& dVar) | 78 |
| bool userPageUnbindXEVENTX(string sPage, string sGraphicalObjectId, bool& bVar) | |
| bool userPageUnbindXEVENTX(string sPage, string sGraphicalObjectId, dio& bVar) | |
| bool userPageUnbindXEVENTX(string sPage, string sGraphicalObjectId, num& nVar) | |
| bool userPageUnbindXEVENTX(string sPage, string sGraphicalObjectId, aio& aVar) | |
| bool userPageUnbindXEVENTX(string sPage, string sGraphicalObjectId, string& sVar) | 79 |
| void userPageSet(string sPage, string sGraphicalObjectId, | |
| string sGraphicalObjectAttrib, bool nVar) | |
| void userPageSet(string sPage, string sGraphicalObjectId, | |
| string sGraphicalObjectAttrib, num nVar) | |
| void userPageSet(string sPage, string sGraphicalObjectId, | |
| string sGraphicalObjectAttrib, string nVar) | 80 |
| void userPageAlert(string sMessage) | 81 |
| bool userPageConfirm(string sMessage) | 81 |
| bool userPagePrompt(string sMessage, string& sEntry) | 81 |
| bool userPagePrompt(string sMessage, num& nNum) | 82 |
| void popUpMsg(string sText), | |
| void popUpMsg(string sText, num nSeverity) | 82 |
| bool logMsg(string sText), | |
| bool logMsg(string sText, num nSeverity) | 82 |
| string getProfile() | 83 |
| num setProfile(string sUserLogin, string sUserPassword) | 83 |
| string getLanguage() | 84 |
| bool setLanguage(string sLanguage) | 84 |
| string getDate(string sFormat) | 85 |

5 - TASKS 87

| | |
|---|-----------|
| 5.1 DEFINITION | 89 |
| 5.2 FORTSETZEN NACH EINEM LAUFZEITFEHLER | 89 |
| 5.3 SICHTBARKEIT | 89 |
| 5.4 SEQUENTIELLES ORDNEN | 90 |
| 5.5 SYNCHRONE TASKS | 91 |
| 5.6 ZEITÜBERLAUF | 91 |

| | |
|--|------------|
| 5.7 AKTUALISIERUNG DER EIN-/AUSGÄNGE | 91 |
| 5.7.1 Benutzerkarten Aktualisierung Standardmodus | 92 |
| 5.7.2 Aktualisierung der Benutzerkarte No Jitter Modus | 92 |
| 5.7.3 Auswahl des Aktualisierungsmodus der Benutzerkarten und Festlegung des Zyklusverhältnisses | 93 |
| 5.8 SYNCHRONISIERUNG | 94 |
| 5.9 GEMEINSAME NUTZUNG VON RESSOURCEN | 95 |
| 5.10 VAL 3 WATCHDOG | 96 |
| bool wdgRefresh() | 96 |
| 5.11 ANWEISUNGEN | 97 |
| void taskSuspend(string sName) | 97 |
| void taskResume(string sName, num nSkip) | 97 |
| void taskKill(string sName) | 98 |
| void setMutex(bool& bMutex) | 98 |
| string help(num nErrorCode) | 98 |
| num taskStatus(string sName) | 99 |
| void taskCreate string sName, num nPriority, program(...) | 100 |
| void taskCreateSync string sName, num nPeriod, bool& bOverrun, program(...) | 101 |
| void wait(bool bCondition) | 102 |
| void delay(num nSeconds) | 102 |
| num clock() | 103 |
| bool watch(bool bCondition, num nSeconds) | 103 |
| 6 - BIBLIOTHEKEN | 105 |
| 6.1 DEFINITION | 107 |
| 6.2 SCHNITTSTELLE | 107 |
| 6.3 KENNUNG DER SCHNITTSTELLE..... | 107 |
| 6.4 INHALT..... | 107 |
| 6.5 VERSCHLÜSSELUNG..... | 108 |
| 6.6 IM SPEICHER LADEN UND AUS DEM SPEICHER ENTFERNNEN | 109 |
| 6.7 ANWEISUNGEN | 111 |
| num identifier:libLoad(string sPath) | 111 |
| num identifier:libLoad(string sPath, string sPassword) | 111 |
| num identifier:libSave(), num libSave() | 111 |
| num libDelete(string sPath) | 111 |
| string identifier:libPath(), string libPath() | 112 |
| bool libList(string sPath, string& sContents[]) | 112 |
| bool identifier:libExist(string sSymbolName) | 112 |
| 7 - BENUTZERTYP..... | 115 |
| 7.1 DEFINITION | 117 |
| 7.2 ERSTELLUNG | 117 |
| 7.3 EINSATZFALL | 117 |

8 - STEUERUNG DES ROBOTERS 119

| | |
|--|------------|
| 8.1 ALLGEMEINE ANWEISUNGEN..... | 121 |
| void disablePower() | 121 |
| void enablePower() | 121 |
| bool isPowered() | 121 |
| bool isCalibrated() | 121 |
| num workingMode(), num workingMode(num& nStatus) | 122 |
| num esStatus() | 122 |
| num getMonitorSpeed() | 123 |
| num setMonitorSpeed(num nSpeed) | 123 |
| string getVersion(string sComponent) | 124 |
| joint getJntRef(string sReferenceName) | 124 |
| 8.2 STROMEINSPARANWEISUNGEN | 125 |
| void hibernateRobot() | 125 |
| void wakeUpRobot() | 125 |
| 8.3 ANWEISUNGEN ZUR VERWALTUNG VON STROMAUSFÄLLEN | 126 |
| bool hasCpuExtPowerSupply() | 126 |
| void prepareCpuShutdown() | 126 |
| 8.4 BREMTESTANWEISUNG | 127 |
| bool brakeTest(num& nBrakeStatus) | 127 |

9 - ARMPOSITION 129

| | |
|---|------------|
| 9.1 EINLEITUNG..... | 131 |
| 9.2 TYP JOINT | 131 |
| 9.2.1 Definition | 131 |
| 9.2.2 Operatoren | 132 |
| 9.2.3 Anweisungen..... | 132 |
| joint abs(joint jPosition) | 132 |
| joint herej() | 133 |
| bool isInRange(joint jPosition) | 133 |
| void setLatch(dio diInput) | 134 |
| bool getLatch(joint& jPosition) | 134 |
| 9.3 TYP TRSF | 135 |
| 9.3.1 Definition | 135 |
| 9.3.2 Orientierung | 136 |
| 9.3.3 Operatoren | 138 |
| 9.3.4 Anweisungen..... | 138 |
| num distance(trsf trPosition1, trsf trPosition2) | 138 |
| trsf interpolateL(trsf trStart, trsf trEnd, num nPosition) | 139 |
| trsf interpolateC(trsf trStart, trsf trIntermediate, trsf trEnd, num nPosition) | 140 |
| trsf align(trsf trPosition, trsf Reference) | 140 |

| | |
|---|------------|
| 9.4 TYP FRAME | 141 |
| 9.4.1 Definition | 141 |
| 9.4.2 Verwendung | 141 |
| 9.4.3 Operatoren | 142 |
| 9.4.4 Anweisungen | 142 |
| num setFrame(point pOrigin, point pAxisOx, point pPlaneOxy, frame& fResult) | 142 |
| trsf position(frame fFrame, frame fReference) | 142 |
| void link(frame fFrame, frame fReference) | 142 |
| 9.5 TYP TOOL..... | 143 |
| 9.5.1 Definition | 143 |
| 9.5.2 Verwendung | 143 |
| 9.5.3 Operatoren | 144 |
| 9.5.4 Anweisungen | 144 |
| void open(tool tTool) | 144 |
| void close(tool tTool) | 145 |
| trsf position(tool tTool, tool tReference) | 145 |
| void link(tool tTool, tool tReference) | 145 |
| 9.6 TYP POINT..... | 146 |
| 9.6.1 Definition | 146 |
| 9.6.2 Operatoren | 146 |
| 9.6.3 Anweisungen | 147 |
| num distance(point pPosition1, point pPosition2) | 147 |
| point compose(point pPosition, frame fReference, trsfl trTransformation) | 147 |
| point appro(point pPosition, trsfl trTransformation) | 148 |
| point here(tool tTool, frame fReference) | 148 |
| point jointToPoint(tool tTool, frame fReference, joint jPosition) | 148 |
| bool pointToJoint(tool tTool, joint jInitial, point pPosition, joint& jResult) | 149 |
| trsfl position(point pPosition, frame fReference) | 149 |
| void link(point pPoint, frame fReference) | 149 |
| 9.7 TYP CONFIG..... | 150 |
| 9.7.1 Einleitung..... | 150 |
| 9.7.2 Definition | 150 |
| 9.7.3 Operatoren | 151 |
| 9.7.4 Konfiguration (Arm RX/TX)..... | 152 |
| 9.7.4.1 Konfiguration der Schulter..... | 152 |
| 9.7.4.2 Konfiguration des Ellenbogens | 153 |
| 9.7.4.3 Konfiguration des Handgelenks | 153 |
| 9.7.5 Konfiguration SCARA (TS/TP Arm)..... | 155 |
| 9.7.6 Anweisungen | 155 |
| config config(joint jPosition) | 155 |

10 - BEWEGUNGSSTEUERUNG 157

| | |
|--|------------|
| 10.1 BEWEGUNGSSTEUERUNG | 159 |
| 10.1.1 Bewegungstypen: Punkt-zu-Punkt, geradlinig, kreisförmig | 159 |
| 10.1.2 Verkettung von Bewegungen: Blending | 161 |
| 10.1.2.1 Blending | 161 |
| 10.1.2.2 Blending aufheben..... | 162 |
| 10.1.2.3 Joint Blending, Kartesisches Blending..... | 162 |
| 10.1.3 Wiederaufnahme einer Bewegung..... | 163 |
| 10.1.4 Besonderheiten der kartesischen Bewegungen (geradlinig, kreisförmig) | 164 |
| 10.1.4.1 Interpolation der Orientierung | 164 |
| 10.1.4.2 Änderung der Konfiguration (Arm RX/TX) | 166 |
| 10.1.4.3 Singularitäten (Arm RX/TX) | 168 |
| 10.2 ANTICIPATION VON BEWEGUNGEN..... | 168 |
| 10.2.1 Prinzip | 168 |
| 10.2.2 Anticipation und Blending..... | 169 |
| 10.2.3 Synchronisierung | 169 |
| 10.3 GESCHWINDIGKEITSSTEUERUNG | 170 |
| 10.3.1 Prinzip | 170 |
| 10.3.2 Einfache Einstellung..... | 170 |
| 10.3.3 Komplexere Einstellungen | 171 |
| 10.3.4 Schleppfehler | 171 |
| 10.4 ONLINE-BEWEGUNGSSTEUERUNG | 171 |
| 10.5 TYP MDESC | 172 |
| 10.5.1 Definition | 172 |
| 10.5.2 Operatoren | 172 |
| 10.6 BEWEGUNGSANWEISUNGEN | 173 |
| num movej(joint jPosition, tool tTool, mdesc mDesc) | 173 |
| num movej(point pPosition, tool tTool, mdesc mDesc) | 173 |
| num movel(point pPosition, tool tTool, mdesc mDesc) | 174 |
| num movec(point pIntermediate, point pTarget, tool tTool, mdesc mDesc) | 175 |
| void stopMove() | 176 |
| void resetMotion(), void resetMotion(joint jStartingPoint) | 176 |
| void restartMove() | 177 |
| void waitEndMove() | 177 |
| bool isEmpty() | 178 |
| bool isSettled(), bool isSettled(tool tTool, num nTransAccuracy), bool isSettled(tool tTool, num nTransAccuracy, num nRotAccuracy, num nTime) | 178 |
| void autoConnectMove(bool bActive), bool autoConnectMove() | 179 |
| num getSpeed(tool tTool) | 179 |
| joint getPositionErr() | 179 |
| void getJointForce(num& nForce) | 179 |
| num getMoveld() | 180 |
| num setMoveld(num nMoveld) | 180 |

| | |
|--|------------|
| 11 - EINSTELLUNG DER ROBOTERNUTZLAST..... | 183 |
| 11.1 PRINZIP..... | 185 |
| 11.2 ANWEISUNGEN | 185 |
| num setPayload(tool tTool, num nMass, trsf trGravityCenter, num& nInertiaMatrix[]) | 185 |
| num getPayload (tool tTool, num& nMass , trsf& trGravityCenter , num& nInertiaMatrix[]) | 186 |
| 12 - SYSTEMEREIGNISSE | 187 |
| void getIds(num& nEvtId) | 189 |
| num getEvents(num& x_nEvtNbr, num& x_nId[], num& x_nType[], string& x_sInfo[]) | 189 |
| 13 - OPTIONEN | 191 |
| 13.1 ALTER: BEWEGUNGSSTEUERUNG IN ECHTZEIT..... | 193 |
| 13.1.1 Prinzip | 193 |
| 13.1.2 Programmierung..... | 193 |
| 13.1.3 Randbedingungen | 193 |
| 13.1.4 Sicherheit | 194 |
| 13.1.5 Begrenzungen | 194 |
| 13.1.6 Anweisungen | 195 |
| num alterMovej(joint jPosition, tool tTool, mdesc mDesc) | 195 |
| num alterMovej(point pPosition, tool tTool, mdesc mDesc) | 195 |
| num alterMovel(point pPosition, tool tTool, mdesc mDesc) | 196 |
| num alterMovec(point plIntermediate, point pTarget, tool tTool, mdesc mDesc) | 196 |
| num alterBegin(frame fAlterReference, mdesc mMaxVelocity) | 197 |
| num alterBegin(tool tAlterReference, mdesc mMaxVelocity) | 197 |
| num alterEnd() | 198 |
| num alter(trsf trAlteration) | 198 |
| num alterStopTime() | 199 |
| 13.2 OEM-LIZENZKONTROLLE | 200 |
| 13.2.1 Grundlagen..... | 200 |
| 13.2.2 Anweisungen | 200 |
| string getLicence(string sOemLicenceName, string sOemPassword) | 200 |
| 13.3 ABSOLUTER ROBOTER | 201 |
| 13.3.1 Prinzip | 201 |
| 13.3.2 Betrieb | 201 |
| 13.3.3 Begrenzungen | 201 |
| 13.3.4 Anweisungen | 202 |
| void getDH (num& theta[], num& d[], num& a[], num& alpha[], num& beta[]) | 202 |
| void getDefaultDH(num& theta[], num& d[], num& a[], num& alpha[], num& beta[]) | 202 |
| bool setDH(num& theta[], num& d[], num& a[], num& b[], num& alpha[], num& beta[]) | 202 |
| 13.4 KONTINUIERLICHE ACHSE..... | 203 |
| 13.4.1 Prinzip | 203 |
| 13.4.2 Anweisungen | 203 |
| joint resetTurn(joint jReference) | 203 |

| | |
|---|------------|
| 14 - ANHANG | 205 |
| 14.1 LAUFZEITFEHLERCODES..... | 207 |
| 14.2 GRAFISCHE OBJEKTATTRIBUTE DER BENUTZERSEITE..... | 210 |
| 15 - ABBILDUNG | 215 |
| 16 - INDEX..... | 217 |

KAPITEL 1

EINLEITUNG

VAL 3 ist eine höhere Programmiersprache für die Steuerung von **Stäubli**-Robotern bei der Ausführung sämtlicher Arten von Anwendungen.

VAL 3 besitzt alle wichtigen Funktionen von gängigen Echtzeit-Programmiersprachen und zusätzlich die speziellen Steuerungsfunktionen für Industrieroboter:

- Steuerungsprogramme des Roboters
- Programme zur Erstellung geometrischer Modelle
- Tools zur Ansteuerung der Ein-/Ausgänge

In dem vorliegenden Handbuch sind alle zur Programmierung notwendigen Begriffe und Anweisungen der **VAL 3**-Sprache eingehend erläutert. Es enthält die Kapitel:

- Sprachelemente
- Einfache Typen
- Benutzerschnittstelle
- Tasks
- Bibliotheken
- Benutzertypen
- Steuerung des Roboters
- Position des Roboterarms
- Bewegungssteuerung

Zum leichteren Auffinden sind die Anweisungen mit ihrer Syntax im Inhaltsverzeichnis angegeben.

VAL 3 Beispiele mit der grafischen Benutzerschnittstelle basieren auf dem Template "VAL3help_example", das mit dem SRC geliefert wird.

KAPITEL 2

ELEMENTE DER SPRACHE VAL 3

Die **VAL 3** Programmiersprache besteht aus Applikationen. Eine **VAL 3**-Applikation besteht aus Programmen und Daten. Eine **VAL 3**-Applikation kann sich auch auf andere Applikationen beziehen, die entweder als Bibliotheken oder als Benutzertyp-Definitionen genutzt werden.

2.1. APPLIKATIONEN

2.1.1. DEFINITION

Eine **VAL 3**-Applikation ist ein eigenständiges Programm zur Steuerung von Robotern und zugehörigen Ein-/Ausgängen mit einem Controller.

Eine **VAL 3**-Applikation besteht aus folgenden Elementen:

- einer Gruppe von **Programmen**: den auszuführenden **VAL 3**-Anweisungen,
- einer Gruppe von **globalen Variablen**: den von allen Programmen der Applikation gemeinsam genutzten Daten
- einer Gruppe von **Bibliotheken**: externe Applikationen zur gemeinsamen Nutzung von Programmen und/oder Daten
- einer Gruppe von **Benutzertypen**: externe Applikationen als Schablonen zur Definition strukturierter Daten in der Applikation
- eine Gruppe von MCP **Benutzerseiten**: Applikation grafische Schnittstelle zur Anzeige auf dem **MCP**

Während ihrer Ausführung enthält eine Applikation außerdem:

- eine Gruppe von Tasks: die gleichzeitig ausgeführten Programme

2.1.2. VORPROGRAMMIERTE INHALTE

Durch Kopieren des Inhalts einer Vorlage (vordefinierte Applikation) wird eine neue **VAL 3**-Applikation erstellt. Neue benutzerspezifische Vorlagen können erstellt werden. Sie bestehen lediglich aus einer standardmäßigen **VAL 3**-Applikation, die in einem dedizierten Verzeichnis des Controllers platziert wird.

Eine **VAL 3**-Applikation kann nur gestartet werden, wenn sie über ein **start()** und ein **stop()** Programm verfügt. Ohne **start()** und **stop()** Programm kann eine **VAL 3**-Applikation nur als Bibliothek oder Definition eines Benutzertyps verwendet werden. Es ist möglich, Applikationen zu definieren, die nur Daten oder nur Programme enthalten.

2.1.3. STARTEN UND BEENDEN DER APPLIKATION

Das Starten einer **VAL 3**-Applikation wird über den Controller gesteuert. Dieses kann entweder manuell von der **MCP**-Oberfläche oder automatisch als Teil des Startvorgangs erfolgen.

Es kann nur jeweils eine **VAL 3**-Applikation gestartet werden. Diese Applikation kann jedoch gleichzeitig viele andere Applikationen (als Bibliotheken) benutzen und viele verschiedene Ausführungstasks starten.

Wenn eine **VAL 3**-Applikation läuft, wird ihr **start()**-Programm ausgeführt.

Wenn die letzte Task beendet ist, schließt die **VAL 3**-Applikation sich selbst: das Programm **stop()** wird dann ausgeführt. Alle eventuell verbleibenden, von den Bibliotheken erstellten Tasks werden in der umgekehrten Reihenfolge ihrer Erstellung gelöscht.

Wird eine **VAL 3**-Applikation von der Benutzerschnittstelle der **MCP** aus unterbrochen, so wird die Start-Task, falls sie noch vorhanden ist, augenblicklich gelöscht. Dann wird das Programm **stop()** ausgeführt, worauf alle noch nicht ausgeführten Tasks der Applikation in der umgekehrten Reihenfolge ihrer Erstellung gelöscht werden.

2.1.4. PARAMETER DER APPLIKATION

Eine **VAL 3**-Applikation wird mit folgenden Parametern konfiguriert:

- Längeneinheit
- Größe des Laufzeitspeichers

Auf diese Parameter kann nicht über eine **VAL 3**-Anweisung zugegriffen werden. Sie können nur über die Oberfläche des **MCP** oder mithilfe vom **VAL 3 Studio** in der **Stäubli Robotics Suite** geändert werden.

2.1.4.1. LÄNGENEINHEIT

In **VAL 3**-Applikationen werden Millimeter oder Inch als Längeneinheit verwendet. Sie dienen zur Beschreibung der geometrischen Daten von **VAL 3**: frame, point, joint (für lineare Achsen), transformation, tool und Überschleifen der Bahn.

Die Längeneinheit wird bei Erstellung einer Applikation durch die im System geltende Längeneinheit festgelegt und kann anschließend nicht mehr geändert werden.

Die Einheiten (mm oder inch) sind eine Eigenschaft der jeweiligen **VAL 3**-Module (Applikationen, Bibliotheken, Definition des Benutzertyps). Die in allen Bibliotheken definierten Einheiten und die in einer Applikation verwendeten Benutzertypen müssen mit den in dieser Applikation definierten Einheiten übereinstimmen.

Bei Inkohärenzen warnt ein Popup-Fenster den Benutzer, dass die Applikation mit inkohärenten Einheiten heruntergeladen wurde.

Wenn die Inkohärenz durch die **VAL 3**-Funktion **libLoad** festgestellt wird, wird die Bibliothek geladen, aber der Alarmcode 1 wird zurückgegeben.

In beiden Fällen enthält das Protokoll Einzelheiten zur festgestellten Inkohärenz.

2.1.4.2. GRÖÙE DES LAUFZEITSPEICHERS

Für jede zu speichernde **VAL 3**-Task wird Speicherplatz benötigt:

- Der Call-Stack (Liste der in dieser Task ausgeführten Programmaufrufe)
- Die Parameter jedes Programms des Call-Stack
- Die lokalen Variablen jedes Programms des Call-Stack

Standardmäßig besitzt jede Task **5000** Byte Laufzeitspeicher. Üblicherweise muss dieser Parameter nicht geändert werden.

Dieser Wert kann jedoch für Applikationen mit besonders großen Array's lokaler Variablen oder rekursiver Algorithmen nicht ausreichen:

In diesem Fall muss er über die Oberfläche des **MCP** oder mithilfe vom **VAL 3 Studio** in **Stäubli Robotics Suite** vergrößert werden oder die Applikation muss durch Verringerung der Anzahl an Programmen im Call-Stack oder durch den Einsatz globaler statt lokaler Variablen optimiert werden.

2.1.4.3. APPLIKATION GRAFISCHE BENUTZERSCHNITTSTELLE (BENUTZERSEITEN)

Zur Anzeige oder Eingabe von Informationen muss die grafische Seite definiert werden. SRS ist zur Konzeption der Benutzerseiten und zur Bindung der grafischen Elemente an die **VAL 3**-Variablen zu verwenden.

Siehe SRS-Dokumentation zu den Bindemechanismen.

2.2. PROGRAMME

2.2.1. DEFINITION

Ein Programm enthält eine Reihe von **VAL 3**-Anweisungen, die auszuführen sind.

Es besteht aus folgenden Elementen:

- Eine Sequenz von **Anweisungen**: den auszuführenden **VAL 3**-Anweisungen,
- einer Gruppe von **lokalen Variablen**: den programminternen Daten,
- einer Gruppe von **Parametern**: den Daten, die dem Programm beim Aufrufen geliefert werden.

Programme dienen zur Zusammenfassung von Anweisungssequenzen, um sie an verschiedenen Stellen in einer Applikation verwenden zu können. Neben dem geringeren Programmierungsaufwand wird die Struktur der Applikationen dadurch vereinfacht, was die Programmierung und Wartung erleichtert und die Lesbarkeit verbessert.

Die Anzahl der Programmanweisungen ist nur durch den im System verfügbaren Speicherplatz begrenzt.

Die Anzahl der lokalen Variablen und Parameter ist nur durch den Laufzeitspeicher der Applikation begrenzt (siehe Kapitel 2.1.4.2).

2.2.2. WIEDEREINSPRUNG

Die Programme erlauben den Wiedereinsprung, d.h. ein Programm kann sich selbst rekursiv aufrufen (**call** Anweisung) oder von mehreren Tasks gleichzeitig aufgerufen werden. Jede Programminstanz verwendet seine eigenen spezifischen lokalen Variablen und Parameter. Zwischen zwei verschiedenen Aufrufen desselben Programms ist keine Interaktion möglich.

2.2.3. PROGRAMM START()

Das Programm **start()** wird zur Ausführung der **VAL 3**-Applikation aufgerufen. Es kann keine Parameter besitzen.

In diesem Programm befinden sich alle zur Ausführung der Applikation erforderlichen Vorgänge: Initialisierung der globalen Variablen, der Ausgänge, Erstellen der Tasks der Applikation u.a.

Die Applikation ist nach Abarbeitung des **start()**-Programms nicht beendet, solange noch andere Tasks der Applikation in Ausführung sind.

Das **start()**-Programm kann, wie jedes andere Programm, in einem anderen Programm aufgerufen werden (Anweisung **call**).

2.2.4. PROGRAMM STOP()

Das Programm **stop()** wird am Ende der Ausführung der **VAL 3**-Applikation aufgerufen. Es kann keine Parameter besitzen.

In diesem Programm finden sich im Allgemeinen alle zum korrekten Beenden der Applikation erforderlichen Vorgänge: Reinitialisierung der Ausgänge, Beenden der Tasks der Applikation in einer bestimmten Reihenfolge u.s.w.

Das **stop()**-Programm kann, wie jedes andere Programm auch, in einem anderen Programm aufgerufen werden (**call**-Anweisung), durch Aufrufen des **stop()** Programms wird die Applikation nicht gestoppt.

2.2.5. ANWEISUNGEN ZUR PROGRAMMSTEUERUNG

Comment //

Syntax

// <String>

Funktion

Eine Zeile, die mit « // » beginnt, wird nicht ausgeführt, das Programm macht mit der nächsten Zeile weiter. « // » können nicht mitten in einer Zeile verwendet werden, sie müssen als Anfangszeichen einer Zeile stehen.

Beispiel

```
// This is an example of a comment
```

call program

Syntax

call program([parameter1][,parameter2])

Funktion

Die call() Anweisung führt ein benutzerdefiniertes Programm aus. Anzahl und Art der Ausdrücke nach dem Programmnamen müssen zur Programmschnittstelle passen. Zuerst werden die als Parameter angegebenen Ausdrücke in der Reihenfolge ihrer Festlegung ausgeführt. Dann werden die lokalen Variablen initialisiert und die Ausführung des Programms startet mit seinen Anfangsanweisungen.

Die Ausführung eines Aufrufs wird beendet, wenn das Programm die Anweisung return oder end ausführt.

Beispiel

```
// Calls the pick() and place() programs for i,j between 1 and 10
for i = 1 to 10
  for j = 1 to 10
    call pick(pPallet1[i,j])
    call place(pPallet2[i,j])
  endFor
endFor
```

return

Syntax

return

Funktion

Die Anweisung return beendet sofort die Ausführung des aktuellen Programms. Wenn das Programm durch einen **call** aufgerufen wurde, wird die Programmausführung im aufrufenden Programm nach dem **call** fortgesetzt. Andernfalls (z. B. wenn das Unterprogramm das Programm **start()** oder der Startpunkt für eine Task ist) wird die laufende Task beendet. Die Anweisung return hat genau dieselbe Wirkung wie die Anweisung end am Ende des Programms.

Programme sind oft einfacher zu verstehen und zu verwalten, wenn ihre Ausführung stets mit der Anweisung end endet. Die Verwendung einer return Anweisung mitten in einem Programm ist also nicht wünschenswert.

if control instruction

Syntax

```

if <bool bCondition>
  <instructions>
[elseif <bool bAlternateCondition1>
  <instructions>]
.|.
[elseif <bool bAlternateConditionN>
  <instructions>]
[else
  <instructions>]
endif

```

Funktion

Die Sequenz **if...elseif...else...endif** bewertet hintereinander die Booleschen Ausdrücke, die **if** oder **elseif** enthalten, bis einer der Ausdrücke zutrifft. Die Anweisungen nach dem Booleschen Ausdruck werden dann bis zum nächsten **elseif**, **else** oder **endif** ausgeführt. Schließlich wird die Ausführung des Programms nach der Anweisung **endif** fortgesetzt.

Wenn alle Booleschen Ausdrücke mit **if** oder **elseif** falsch sind, werden die Anweisungen zwischen **else** und **endif** ausgeführt (wenn **else** vorkommt). Die Ausführung des Programms wird dann nach der Anweisung **endif** fortgesetzt.

Die Anzahl an **elseif**-Ausdrücken innerhalb einer **if...endif**-Sequenz ist nicht beschränkt.

Die Sequenz **if...elseif...else...endif** kann durch die Sequenz **switch...case...default...endSwitch** ersetzt werden, wenn die verschiedenen möglichen Werte eines einzelnen Ausdrucks getestet werden.

Beispiel

Dieses Programm wandelt einen in eine **string**-Variable (**sDay**) geschriebenen Tag in ein **num** (**nDay**).

```

sOutput="Enter a day"
sInput=sDay
if sDay=="Monday"
  nDay=1
elseif sDay=="Tuesday"
  nDay=2
elseif sDay=="Wednesday"
  nDay=3
elseif sDay=="Thursday"
  nDay=4
elseif sDay=="Friday"
  nDay=5
else
  // Weekend !
  nDay=0
endif

```

Siehe auch

switch control instruction

while control instruction

Syntax

```
while <bool bCondition>
  <instructions>
endWhile
```

Funktion

Die Anweisungen zwischen **while** und **endWhile** werden solange ausgeführt, wie die boolesche Bedingung **bCondition** wahr (**true**) ist.

Wenn die boolesche Bedingung **bCondition** bei der ersten Überprüfung nicht wahr ist, werden die Anweisungen zwischen **while** und **endWhile** nicht ausgeführt.

Parameter

| | |
|------------------------|--------------------------------------|
| bool bCondition | zu überprüfender boolescher Ausdruck |
|------------------------|--------------------------------------|

Beispiel

```
// This simple program makes a signal flash as long as the robot is moving
diLamp = false
while (isSettled() == false)
  // Inverses the value of the diLamp: true false
  diLamp = !diLamp
  // Waits ½ s
  delay(0.5)
endWhile
diLamp = false
```

do ... until control instruction

Syntax

```
do
  <instructions>
until <bool bCondition>
```

Funktion

Die Anweisungen zwischen **do** und **until** werden solange ausgeführt, bis die boolesche Bedingung **bCondition** wahr (**true**) ist.

Die Anweisungen zwischen **do** und **until** werden einmal ausgeführt, wenn die boolesche Bedingung **bCondition** bei der erstmaligen Überprüfung true ist.

Parameter

| | |
|------------------------|--------------------------------------|
| bool bCondition | zu überprüfender boolescher Ausdruck |
|------------------------|--------------------------------------|

for control instruction

Syntax

```
for <num nCounter> = <num nBeginning> to <num nEnd> [step <num nStep>]
  <instructions>
endFor
```

Funktion

Die Anweisungen zwischen **for** und **endFor** werden solange ausgeführt, bis **nCounter** den angegebenen Wert **nEnd** überschreitet.

Der **nCounter** wird auf den Wert **nBeginning** initialisiert. Wenn **nBeginning** **nEnd** überschreitet, werden die Anweisungen zwischen **for** und **endFor** nicht ausgeführt. Bei jedem Iterationsschritt wird **nCounter** um den Wert **nStep** und die Anweisungen zwischen **for** und **endFor** werden, solange der **nCounter** den Wert **nEnd** nicht überschreitet, wiederholt.

Wenn **nStep** positiv ist, stoppt die **for**-Schleife wenn **nCounter** größer ist als **nEnd**. Wenn **nStep** negativ ist, stoppt die **for**-Schleife wenn **nCounter** kleiner ist als **nEnd**.

Parameter

| | |
|-----------------------|--|
| num nCounter | Variable des Typs num , die als Zähler verwendet wird |
| num nBeginning | Numerischer Ausdruck zur Initialisierung des Zählers |
| num nEnd | Numerischer Ausdruck zum Test des Schleifenendes |
| [num nStep] | Numerischer Ausdruck zur Erhöhung des Zählers |

Beispiel

```
// This program rotates axis 1 from -90° to +90° in -10° steps
for nPos = 90 to -90 step -10
  jDest.j1 = nPos
  movej(jDest, flange, mNomSpeed)
  waitEndMove()
endFor
```

switch control instruction

Syntax

```
switch <expression>
case <value1> [, <value2>]
  <instructions1-2>
  break
[case <value3> [, <value4>]
  <instructions3-4>
  break ]
[default
  <Default Instructions>
  break ]
endSwitch
```

Funktion

Die Sequenz **switch...case...default...endSwitch** bewertet hintereinander die Ausdrücke, die **case** enthalten, bis einer der Ausdrücke nach dem **switch** dem ursprünglichen Ausdruck entspricht.

Die Anweisungen nach dem Ausdruck werden dann bis zum **break** ausgeführt. Schließlich wird die Ausführung des Programms nach der Anweisung **endSwitch** fortgesetzt.

Wenn kein **case**-Ausdruck mit dem anfänglichen **switch**-Ausdruck gleichwertig ist, werden die Anweisungen zwischen **default** und **endSwitch** ausgeführt (wenn **default** vorkommt).

Die Anzahl an **case**-Ausdrücken innerhalb einer **switch...endSwitch**-Sequenz ist nicht beschränkt. Die Ausdrücke nach **case** müssen vom selben Typ sein wie der Ausdruck nach **switch**.

Die Sequenz **switch...case...default...endSwitch** ist der Sequenz **if...elseif...else...endif** sehr ähnlich. Sie akzeptiert nicht nur Boolesche Ausdrücke, sondern Ausdrücke jeglichen Typs, die den standardmäßigen Operator "is equal to" "==" unterstützen.

Beispiel

Dieses Programm bewertet eine **num**-Variable (**nMenu**) die einem Tastendruck entspricht und ändert den **string** **s** entsprechend.

```
nMenu = sInput
switch nMenu
  case 271
    s = "Menu 1"
  break
  case 272
    s = "Menu 2"
  break
  case 273, 274, 275, 276, 277, 278
    s = "Menu 3 to 8"
  break
  default
    s = "this key is not a menu key"
  break
endSwitch
```

Dieses Programm wandelt einen in eine **string**-Variable (**sDay**) geschriebenen Tag in ein **num** (**nDay**).

```
sOutput="Enter a day: "
sDay=sInput
switch sDay
    case "Monday"
        nDay=1
    break
    case "Tuesday"
        nDay=2
    break
    case "Wednesday"
        nDay=3
    break
    case "Thursday"
        nDay=4
    break
    case "Friday"
        nDay=5
    break
default
    // Not a week day !
    nDay=0
break
endswitch
```

2.3. DATENELEMENTE

2.3.1. DEFINITION

Ein Datenelement ist ein Satz von Werten, die als Parameter verwendet oder als Ergebnis von **VAL 3**-Anweisungen verwendet werden.

Datenelemente bestehen aus:

- eine bestimmte Anzahl von Werten
- einem Typ, durch den die möglichen Werte und erlaubten Vorgänge für die Datenelemente definiert werden. Boolesche, numerische und Strings sind die einfachsten Datentypen
- einem Container, der die Art und Weise definiert, auf welche die Daten gespeichert werden. Die möglichen Datencontainer in **VAL 3** lauten Element, Array und Collection

2.3.2. EINFACHE TYPEN

Die Sprache **VAL 3** unterstützt folgende einfache Typen:

- Typ **bool**: für boolesche Werte (true/false)
- Typ **num**: für numerische Werte (ganzzahlig oder Gleitkomma)
- Typ **string**: für Zeichenketten (Unicode-Zeichen)
- Typ **dio**: für digitale Ein- und Ausgänge
- Typ **aio**: für numerische Ein-/Ausgänge (analog oder digital)
- Typ **sio**: für serielle Ein-/Ausgänge und Ethernet Sockets

In der Dokumentation wird der Variablenotyp durch den ersten Buchstaben seines Namens in Kleinbuchstaben angegeben.:

- **bVariable** ist eine Variable des Typs **bool**
- **nVariable** ist eine Variable des Typs **num**
- **sVariable** ist eine Variable des Typs **string**
- **diVariable** ist eine Variable des Typs **dio**
- **aiVariable** ist eine Variable des Typs **aio**
- **siVariable** ist eine Variable des Typs **sio**

2.3.3. STRUKTURIERTE TYPEN

Ein strukturierter Typ kombiniert mehrere einfachere Typen in einem neuen Typ auf höherem Niveau. Jeder Untertyp erhält einen Namen und ist als Feld der Struktur individuell zugänglich. Adäquate Typen in einer Applikation organisieren Daten so, dass Rechenvorgänge und Programmentwicklungen erleichtert werden.

Die **VAL 3**-Sprache unterstützt folgende aus einfachen Typen erstellte strukturierte Typen:

- Typ **trsf**: für kartesische Koordinatentransformationen
- Typ **frame**: für das kartesische Koordinatensystem
- Typ **tool**: für die am Roboter montierten Tools
- Typ **point**: für die kartesischen Toolpositionen
- Typ **joint**: für die Positionen von Roboterachsen
- Typ **config**: für die Roboterkonfigurationen
- Typ **mdesc**: für die Bewegungsparameter des Roboters

Die **VAL 3**-Sprache unterstützt auch Benutzertypen, die einfache oder strukturierte **VAL 3** oder sogar andere Benutzertypen in einem neuen Typ kombinieren. Ein Benutzertyp kann in einer Applikation als standardmäßiger Typ verwendet werden.

In der Dokumentation wird der Variablenotyp durch den ersten Buchstaben seines Namens in Kleinbuchstaben angegeben.:

- **trVariable** ist eine Variable des Typs **trsf**
- **fVariable** ist eine Variable des Typs **frame**
- **tVariable** ist eine Variable des Typs **tool**
- **pVariable** ist eine Variable des Typs **point**

- **jVariable** ist eine Variable des Typs **joint**
- **cVariable** ist eine Variable des Typs **config**
- **mVariable** ist eine Variable des Typs **mdesc**

2.3.4. CONTAINER

Der Datencontainer definiert, wie die Werte im Datenelement gespeichert werden:

- Ein 'Element'-Container besteht einfach aus einem einzigen Wert. True (Boolean), 0 (Numerisch), 'Text' (String) haben einen Element-Container.
- Ein 'Array'-Container besteht aus einem Wertesatz, der durch 1, 2 oder 3 ganze Zahlen identifiziert wird. Der Startindex in Arrays ist stets 0.
- Ein 'Collection'-Container besteht aus einem Wertesatz, der durch einen String-Schlüssel identifiziert wird. Sämtliche nicht leeren Strings können als Werte-Bezeichner dienen.

Ein eindimensionaler Array-Container mit einem einzelnen Wert (Index 0) wird als Element-Container betrachtet.

In der Dokumentation wird der Container der Variablen bei Bedarf mit dem Namen der Variablen bezeichnet:

- **s1dArray** ist ein eindimensionales Array des Typs **string**
- **s2dArray** ist ein zweidimensionales Array des Typs **string**
- **s3dArray** ist ein dreidimensionales Array des Typs **string**
- **sColl** ist eine Collection des Typs **string**

Einige Anweisungen (für den Umgang mit Arrays oder Collections) berücksichtigen den Datentyp nicht. In der Dokumentation wird der Typ dann durch einen Stern ersetzt: '*'.

2.4. DATENINITIALISIERUNG

2.4.1. DATEN EINFACHEN TYPEN

Die genaue Syntax von Daten einfachen Typen ist im betreffenden Kapitel der einfachen Typen angegeben. Arrays oder Collections müssen Element für Element initialisiert werden. Der Initialisierungswert lautet **false** für bool, **0** für num und **""** (leerer String) für einen String.

Beispiel

In diesem Fall ist bBool eine Boolesche, nPi eine numerische und sString eine String-Variable.

```
bBool = true
nPi = 3.141592653
sString = "this is a string"
```

2.4.2. DATEN STRUKTURIERTEN TYPEN

Der Wert von Daten strukturierten Typs wird durch die Sequenz ihrer Feldwerte zwischen den {} Klammern, durch Kommata getrennt, definiert. Leere Felder werden durch 0 ersetzt. Die Reihenfolge ist im Kapitel über die strukturierte Typen angegeben. Der Wert einer Struktur kann Werte anderer, verschachtelter Unterstrukturen enthalten. Arrays oder Zusammenstellungen eines strukturierten Typs müssen Element für Element initialisiert werden.

Beispiel

Der Typ Point besteht aus einer trsf und einem Konfigurationstyp. Eine Point-Variable kann wie gezeigt initialisiert werden:

```
pPosition = {{100, -50, 200, 0, 0, 0}, {sfree, efree, wfree}}
```

Die Transformation des Point kann auch folgendermaßen initialisiert werden:

```
pPosition.trsf = {100, -50, 200, , ,}
```

2.5. VARIABLEN

2.5.1. DEFINITION

Eine Variable ist ein Datenelement in einer Applikation oder einem Programm, das durch seinen Namen gekennzeichnet ist.

Eine Variable ist definiert durch:

- Namen: eine Zeichenkette
- einen Gültigkeitsbereich: von wo auf die Variable zugegriffen werden kann (innerhalb eines einzelnen Programms, gemeinsam durch Programme innerhalb einer Applikation oder gemeinsam durch verschiedene Applikationen)
- eine bestimmte Anzahl von Werten
- einen Datentyp (einfacher oder strukturierter Typ)
- einen Daten-Container (Element, Array oder Collection)

Ein Variablenname ist eine aus "**a..zA..Z0..9_**" ausgewählte Zeichenkette, die mit einem Buchstaben beginnt.

2.5.2. GÜLTIGKEITSBEREICH EINER VARIABLEN

Der Gültigkeitsbereich der Variablen ist folgendermaßen:

- global: alle Programme der Anwendung können die Variable verwenden, oder
- lokal: diese Variable kann nur mit dem Programm verwendet werden, in dem sie deklariert wurde

Haben eine globale und eine lokale Variable den gleichen Namen, wird das Programm, in dem die lokale Variable deklariert ist, die lokale Variable verwenden und nicht auf die globale Variable zugreifen.

Wenn eine Applikation als Bibliothek verwendet wird, kann jede globale Variable entweder als public oder private eingestuft werden. Auf eine Variable des Typs public können Applikationen zugreifen, die die Bibliothek verwenden, der Zugriff auf Variablen des Typs private ist nur innerhalb der Bibliothek möglich.

2.5.3. ZUGRIFF AUF DEN WERT EINER VARIABLEN

Der Zugriff auf den Wert einer Variablen ist von deren Container abhängig:

- Der Zugriff auf den Wert eines Element-Containers erfolgt mit dem Namen der Variablen (ohne eckige Klammern): nVariable.
- Der Zugriff auf den Wert in einem Array erfolgt mit seinen numerischen Indizes in eckigen Klammern hinter dem Namen der Variablen: n1dArray[nIndex], n2dArray[nIndex1, nIndex2], n3dArray[nIndex1, nIndex2, nIndex3].
- Der Zugriff auf den Wert in einer Collection erfolgt mit seinem String-Schlüssel in eckigen Klammern hinter dem Namen der Variablen: nCollection[sKey]

Ein eindimensionaler Array-Container mit einem einzelnen Wert (Index 0) wird als Element-Container betrachtet. Der Zugriff auf seinen Wert ist ohne Klammern möglich: n1dArray entspricht n1dArray[0].

Die zum Zugriff auf einen Wert in einem Array verwendeten numerischen Indizes werden auf den nächsten ganzzahligen Wert gerundet: n2dArray[5.01, 6.99] entspricht n2dArray[5, 7]

Der zum Zugriff auf einen Wert in einem Array verwendete Index liegt zwischen 0 und der Größe der Dimension minus eins.

Die Felder einer strukturierten Variablen sind mit einem '.' gefolgt vom Namen des Feldes zugänglich: pPoint.trsf.x bezieht sich auf den Wert des 'x' Felds des 'trsf' Felds der Point-Daten pPoint.

Beispiel

Initialisierung einfacher Variablen mit verschiedenen Containern:

```
nPi = 3.141592653
sMonth[0] = "January"
sProductName["D 243 064 40 A"] = "VAL 3 CdRom"
```

2.5.4. FÜR ALLE VARIABLENTYPEN GELTENDE BEFEHLE

num **size(*)**

Funktion

Die Anweisung überträgt die Anzahl der zugänglichen Werte innerhalb der Variablen:

- die Größe der Variablen eines Element-Containers ist 1.
- die Größe eines eindimensionalen Arrays entspricht der Anzahl an Elementen im Array.
- die Größe einer Collection entspricht der Anzahl an Elementen in der Collection.

Für zwei- und dreidimensionale Arrays verlangt die size Anweisung einen zweiten Parameter zur Angabe der abzufragenden Dimension. Für ein eindimensionales Array entspricht `size(s1dArray)` `size(s1dArray, 1)`.

Die Größe eines eindimensionalen Arrays übergeben per "Übergabe bei Reference" ist von dem beim Aufruf des Programms angegebenen Index abhängig. Nur der vom angegebenen Index ausgehende Teil des Array ist innerhalb des Unterprogramms zugänglich: `size(s1dArray[nIndex]) = size(s1dArray) - nIndex`.

Parameter

| | |
|-----------------|--------------------------|
| variable | Variable beliebigen Typs |
|-----------------|--------------------------|

Beispiel

Die abzufragende Variable muss ohne eckige Klammern angegeben werden:

```
// Ok
nNbElements=size(sCollection)
// compilation error: unexpected key
nNbElements=size(sCollection[sKey])
```

Für ein eindimensionales Array wird durch einen Index der Start eines Unter-Array angegeben: `size(s1dArray[nIndex])` ist die Größe des mit dem Index nIndex startenden Unter-Arrays.

bool **isDefined(*)**

Funktion

Diese Anweisung gibt `true` zurück, wenn das angegebene Element in einem Array oder einer Collection definiert ist bzw. `false` wenn das Element nicht definiert wurde.

Er kann benutzt werden, um zu testen, ob ein Element in einer Collection definiert ist; außerdem kann sie benutzt werden, um zu testen, ob eine Bibliothek eine Variable ihrer Schnittstelle verwendet oder nicht. Dies ist hilfreich zur Weiterentwicklung der Schnittstelle einer Bibliothek und erlaubt eine angepasste Verarbeitung je nach Versionsstand der Schnittstelle.

Beispiel

In diesem Beispiel wird ein neuer Artikel-Schlüssel in eine Collection aufgenommen.

```
// Get reference name from user input
sReference=sInput
if isDefined(sReferenceColl[sReference])==true
  sOutput="Fehler: reference already defined"
else
  // Add new article in the collection
  insert(sReferenceColl[sReference])
endif
```

Dieses Beispiel testet die Schnittstelle einer Bibliothek.

```
// Load part library
nLoadCode = part:libLoad(sPartPath)
// part:sVersion was not defined in the first version of the library
// Test if this library defines it
if (nLoadCode==0) or (nLoadCode==11)
  if (isDefined(part:sVersion)==false)
    // initial version
    sLibVersion = "v1.0"
  else
    sLibVersion = part:sVersion
  endIf
endIf
```

bool insert(*)

Funktion

Diese Anweisung erzeugt einen neuen Wert des VariablenTyps und speichert ihn im Variablencontainer. Der neue Wert wird mit dem Standardwert des Typs initialisiert. Die Größe der Variablen wird um eins erhöht.

Für ein eindimensionales Array wird der neue Wert an der übergebenen Indexposition eingefügt. Die Indexposition kann der Array-Größe entsprechen: In diesem Fall erfolgt das Einfügen am Ende des Arrays. "insert(s1dArray[size(s1dArray)])" entspricht "append(s1dArray)*".

Für Collections ist das Einfügen nur möglich, wenn der Schlüssel nicht schon in der Collection verwendet wird. Der neue Wert wird mit dem angegebenen Schlüssel verknüpft und die Funktion gibt `true` zurück. Die Anweisung hat keinen Einfluss und gibt `false` zurück, wenn der Schlüssel schon verwendet wurde. Die Anweisung `isDefined()` kann benutzt werden, um zu prüfen, ob ein Schlüssel in einer Collection verwendet wird oder nicht.

Diese Anweisung wird für zwei- und dreidimensionale Arrays (stattdessen Anweisung `resize()` verwenden) sowie für lokale Array-Variablen nicht unterstützt. Die Größe einer Variablen ist auf 9999 Werte begrenzt. Ein Laufzeitfehler entsteht, wenn die Größe der Variablen diese Grenze überschreitet.

Die `insert()`-Anweisung reserviert Systemspeicher. Die Performance der Speicherzuteilung ist nicht garantiert. Deshalb sollte eine ausgiebige Verwendung von `insert()` für **VAL 3**-Applikationen bei Leistungsproblemen vermieden werden.

Beispiel

In diesem Beispiel wird ein neuer Artikel in eine Liste eingefügt.

```
// sInput in the format articleName,position
nPos=find(sInput,",")
// Define new article name
sArticleName=left(sInput,nPos)
// Define the position in the list
toNum(left(sInput,nPos), nIndex, bOk)
if (nIndex<0) or (nIndex>size(sArticleList))
  sOutput="Fehler: invalid position"
else
  // Add new article in the list
  insert(sArticleList[nIndex])
  sArticleList[nIndex] = sArticleName
endif
```

In diesem Beispiel wird ein neuer Artikel-Schlüssel in eine Collection aufgenommen.

```
// Ask for a new article name
sArticleName=sInput
if isDefined(sArticleColl[sArticleName])==true
  sOutput="Fehler: reference already defined"
else
  // Ask new article in the collection
  insert(sArticleColl[sArticleName])
endif

bool delete(*)
```

Funktion

Diese Anweisung löscht den angegebenen Wert aus der Collection. Die Größe der Variablen wird um eins verringert.

Liegt der angegebene Index oder Schlüssel nicht im Bereich wird ein Laufzeitfehler generiert. Die Anweisung `isDefined()` kann benutzt werden, um zu prüfen, ob ein Schlüssel in einer Collection verwendet wird oder nicht.

Die Größe einer Collection kann null betragen, aber eine Array-Variable muss stets mindestens ein Element besitzen. Ein Laufzeitfehler entsteht, wenn versucht wird, das letzte Element eines Arrays zu löschen.

Diese Anweisung wird für zwei- und dreidimensionale Arrays (stattdessen Anweisung `resize()` verwenden) sowie für lokale Array-Variablen nicht unterstützt.

Die `delete()`-Anweisung gibt Systemspeicher frei. Die Performance bei der Freigabe von Systemspeicher ist nicht garantiert. Deshalb sollte eine ausgiebige Verwendung von `delete()` für **VAL 3**-Applikationen bei Leistungsproblemen vermieden werden.

Beispiel

In diesem Beispiel wird ein Artikel in einer Collection gelöscht.

```
// Define article to delete
sArticleName=sInput
if isDefined(sArticleColl[sArticleName])==true
  // remove the article from the collection
  delete(sArticleColl[sArticleName])
else
  sOutput="Fehler: article not defined"
endif
```

num **getData(string sDataName, *)**

Funktion

Diese Anweisung kopiert den Wert der durch den String **sDataName** angegebenen Variable in die spezifizierte Variable. Sind sowohl die Daten als auch Variable eindimensionale Arrays, werden durch die Anweisung **getData()** die Werte aller Array-Einträge kopiert, bis das Ende eines Arrays erreicht ist. Die Anweisung überträgt die Anzahl der kopierten Werte.

Der Datenname muss folgendes Format haben: "library:name[index]", wobei "library:" und "[index]" optional sind:

- "name" ist der Name der Variable
- "library" ist der Name der Bibliotheks kennung in der die Daten definiert sind
- "index" ist der numerische Wert des Index, auf den zugegriffen werden muss, wenn es sich bei den Daten um ein eindimensionales Array handelt

Die Anweisung überträgt einen Fehlercode, wenn die Datenkopie nicht erfolgen konnte:

| Übertragener Wert | Beschreibung |
|-------------------|---|
| n > 0 | Die Variable wurde erfolgreich aktualisiert und n Werte kopiert |
| -1 | Die Daten existieren nicht |
| -2 | Bibliotheks kennung existiert nicht |
| -3 | Index außerhalb des zulässigen Bereichs |
| -4 | Daten- und Variablentyp stimmen nicht überein |

Beispiel

Dieses Programm verbindet 2 Punkt-Arrays pApproach[] und pTrajectory[] aus einer Bibliothek, in einen einzigen lokalen Array pPath[].

```
// Copy approach points in path
i = getData("Part:pApproach", pPath)
if(i > 0)
nPoints = i
// Append trajectory points in path
i = getData("Part:pTrajectory", pPath[nPoints])
if(i >0)
nPoints=nPoints+i
endif
endif
```

2.5.5. FÜR DIE ARRAY-VARIABLEN SPEZIFISCHE ANWEISUNGEN

void append(*)

Funktion

Diese Anweisung erzeugt einen neuen Wert des Variabtentyps und speichert ihn am Ende der eindimensionalen Array-Variablen. Der neue Wert wird mit dem Standardwert des Typs initialisiert. Die Größe der Variablen wird um eins erhöht.

Diese Anweisung wird für zwei- und dreidimensionale Arrays sowie für lokale Array-Variablen nicht unterstützt. Die Größe einer Variablen ist auf 9999 Werte begrenzt. Ein Laufzeitfehler entsteht, wenn die Größe der Variablen diese Grenze überschreitet.

Die append()-Anweisung weist Systemspeicher zu. Die Performance der Speicherzuteilung ist nicht garantiert. Deshalb sollte eine ausgiebige Verwendung von `append()` für **VAL 3**-Applikationen bei Leistungsproblemen vermieden werden.

Beispiel

In diesem Beispiel wird ein neuer Artikel in einer Liste angehängt.

```
// Ask for a new article name
sArticle= sInput
append(sArticleList)
sArticleList[size(sArticleList)-1] = sArticle
```

num size(*, num nDimension)

Funktion

Diese Anweisung gibt die Größe der angegebenen Dimension im Array zurück. Falls nDimension die Dimensionen des Arrays übersteigt, gibt die Funktion 0 zurück. Für ein eindimensionales Array entspricht `size(s1dArray, 1)` `size(s1dArray)`.

Beispiel

Die Array-Variable muss ohne eckige Klammern angegeben werden:

```
//Ok
nNbElements=size(s3dArray,3)
// Compilation error: unexpected indices
nNbElements=size(s3dArray[1,2,3],3)
```

void resize(*, num nDimension, num nSize)

Funktion

Diese Anweisung erzeugt oder löscht Werte in einem Array, wodurch die Größe der angegebenen Dimension dem Wert nSize entspricht. Das Erzeugen und Löschen von Werten erfolgt am Ende des Arrays. Die eventuellen neuen Werte werden mit dem Standardwert des Typs initialisiert.

Diese Anweisung wird für lokale Array-Variablen nicht unterstützt. Die Größe einer Variablen ist auf 9999 Werte begrenzt. Ein Laufzeitfehler entsteht, wenn die Größe der Variablen diese Grenze überschreitet.

Die `resize()`-Anweisung weist Systemspeicher zu oder gibt ihn frei. Die Performance der Speicherzuteilung ist nicht garantiert. Deshalb sollte eine ausgiebige Verwendung von `resize()` für **VAL 3**-Applikationen bei Leistungsproblemen vermieden werden.

Beispiel

Die Variable muss ohne Index angegeben werden. Die folgende Anweisung ändert s2dArray so, dass die zweite Dimension die Größe 5 erhält.

```
resize(s2dArray, 2, 5)
```

2.5.6. SPEZIFISCHE ANWEISUNGEN FÜR COLLECTION-VARIABLEN

string first(*)

Funktion

Diese Anweisung gibt den ersten Schlüssel einer Collection, sortiert in alphabetischer Reihenfolge, zurück. Ist die Collection leer, gibt die Anweisung einen leeren String "" zurück.

string next(*)

Funktion

Diese Anweisung gibt den nächsten Schlüssel einer Collection, sortiert in alphabetischer Reihenfolge, zurück. Handelt es sich bei dem angegebenen Schlüssel um den Letzten in der Collection, gibt die Anweisung einen leeren String "" zurück.

Beispiel

In diesem Beispiel werden alle Elemente einer Collection in alphabetischer Reihenfolge der Schlüssel geordnet auf der Benutzerseite angezeigt:

```
sKey = first(sCollection)
while sKey != ""
    sKey = next(sCollection[sKey])
    sOutput=sKey
endWhile
```

string last(*)

Funktion

Diese Anweisung gibt den letzten Schlüssel einer Collection, sortiert in alphabetischer Reihenfolge, zurück. Ist die Collection leer, gibt die Anweisung einen leeren String "" zurück.

string prev(*)

Funktion

Diese Anweisung gibt den vorherigen Schlüssel einer Collection in alphabetischer Reihenfolge sortierter Schlüssel zurück. Handelt es sich bei dem angegebenen Schlüssel um den Ersten in der Collection, gibt die Anweisung einen leeren String "" zurück.

Beispiel

In diesem Beispiel werden alle Elemente einer Collection in umgekehrter alphabetischer Reihenfolge der Schlüssel geordnet auf der Benutzerseite angezeigt:

```
sKey = last(sCollection)
while sKey != ""
    sKey = prev(sCollection[sKey])
    sOutput=sKey
endWhile
```

2.6. PROGRAMMPARAMETER

Unterprogrammparameter sind Daten, die mit der Aufrufanweisung von einem aufrufenden Programm an ein Unterprogramm übergeben werden. In dem Unterprogramm verhalten sich Parameter wie lokale Variablen, die beim Start des Unterprogramms automatisch initialisiert werden.

Es gibt verschiedene Arten, eine Variable an ein Unterprogramm zu übergeben:

- Sie möchten eventuell nur einen Wert (ein Element) der Variablen übertragen oder den Container (Array oder Collection) der Variablen als Ganze.
- Sie möchten dem Unterprogramm erlauben den Wert der Variablen zu ändern (Variable "by Reference" übertragen) oder lediglich eine Kopie der Variablen an das Unterprogramm zu übertragen, um zu gewährleisten, dass die Variable unverändert bleibt (Variable "by Value" übertragen).

Eine Variable kann als Parameter übergeben werden:

- als Elementwert.
- als Elementreferenz.
- als Array- oder Collection-Referenz.

Übergabe eines Containers (Array oder Collection) als Wert ist verboten.

Eine Referenz wird mit dem '&' Symbol hinter dem Datentyp in der Schnittstellendefinition des Programms gekennzeichnet:

`num& nData` ist ein per Elementreferenz übergebener num Parameter.

`num& n1dArray[]` ist ein per Array-Referenz (eindimensionales Array) übergebener num Parameter.

`num& n2dArray[,]` ist ein per Array-Referenz (zweidimensionales Array) übergebener num Parameter.

`num& n3dArray[,,]` ist ein per Array-Referenz (dreidimensionales Array) übergebener num Parameter.

`num& nCollection[""]` ist ein per Collection-Referenz übergebener num Parameter.

In der Dokumentation wird zur Beschreibung der Anweisung die gleiche Notation verwendet:

`bool pointToJoint(tool tTool, joint jInitial, point pPosition, joint& jResult)` ist eine Anweisung, die einen booleschen Wert zurückgibt und dazu ein per Elementwert übertragenes Tool-, Joint- und Point-Datenelement benutzt sowie einen per Elementreferenz übertragenen Joint.

`num fromBinary(num& nDataByte[], num nDataSize, string sDataFormat, num& nValue[])` ist eine Anweisung, die einen numerischen Wert zurückgibt und dazu ein eindimensionales Array als ersten Parameter (per Referenz übertragen), ein per Elementwert übertragenes numerisches, ein String-Datenelement sowie ein eindimensionales Array als letzten Parameter (per Elementreferenz übertragen) benötigt.

2.6.1. PARAMETER PER ELEMENTWERT

Bei einer Parameter-Definition per Elementwert erstellt das System eine lokale Variable und initialisiert diese mit dem Wert der **VAL 3**-Anweisung, der vom aufrufenden Programm geliefert wird. Ist die gelieferte Anweisung eine Variable, so wird der Parameter mit einer Kopie des Werts der Variablen initialisiert. Alle im Unterprogramm vorgenommenen Änderungen am Wert des Parameters haben keine Auswirkung auf den Wert der Variablen im Aufrufprogramm.

Beispiel

`sendMessage(string sMessage)` ist ein Programm mit einem per Elementwert übertragenen Einzelparameter.

`sendMessage()` kann mit einem konstanten Datenelement oder dem Ergebnis einer Berechnung genutzt werden:

```
call sendMessage("Waiting for signal StartCycle")
call sendMessage("Waiting for signal"+sSignalName)
```

`sendMessage()` kann mit Werten aus Elementen, Arrays oder Collections genutzt werden:

```
call sendMessage(sMessage)
call sendMessage(sMessageArray[23])
call sendMessage(s2dArray[12,3])
call sendMessage(s3dArray[5,7,9])
call sendMessage(sMessageColl[sMessageName])
```

Nach diesen Aufrufen hat sich der Wert von `sMessage`, `sMessageArray[23]`, `s2dArray[12,3]`, `s3dArray[5,7,9]`, `sMessageColl[sMessageName]` durch die Anweisungen in `sendMessage()` nicht geändert.

2.6.2. PARAMETER PER ELEMENTREFERENZ

Bei einer Parameter-Definition per Elementreferenz erstellt das System eine lokale Variable und initialisiert diese mit einem Link zu den Daten, die vom aufrufenden Programm geliefert werden. Die per Referenz übertragene Variable kann ein Element-, Array- oder Collection-Container sein, aber nur der im Aufruf angegebene Wert wird dem Unterprogramm übergeben. Der Container des Parameters ist stets ein Element. Alle am Wert des Parameters im Unterprogramm vorgenommenen Änderungen wirken sich direkt auf den entsprechenden Wert der Daten des Aufrufprogramm aus. **VAL 3** Konstantendaten oder das Ergebnis eines **VAL 3**-Ausdrucks können nicht per Elementreferenz übertragen werden.

Beispiel

`sendMessage(string& sMessage)` ist ein Programm mit einem per Elementreferenz übertragenen Einzelparameter.

`sendMessage()` kann nicht mit einem konstanten Datenelement oder dem Berechnungsergebnis benutzt werden:

```
// compilation errors: variable expected as parameter
call sendMessage("Waiting for signal StartCycle")
call sendMessage("Waiting for signal"+sSignalName)
```

`sendMessage()` kann mit Werten aus Elementen, Arrays oder Collections genutzt werden:

```
call sendMessage(sMessage)
call sendMessage(sMessageArray[23])
call sendMessage(s2dArray[12,3])
call sendMessage(s3dArray[5,7,9])
call sendMessage(sMessageColl[sMessageName])
```

Nach diesen Aufrufen kann sich der Wert von `sMessage`, `sMessageArray[23]`, `s2dArray[12,3]`, `s3dArray[5,7,9]`, `sMessageColl[sMessageName]` durch die Anweisungen in `sendMessage()` geändert haben.

2.6.3. PARAMETER PER ARRAY- ODER COLLECTION-REFERENZ

Bei einer Parameter-Definition per Array- oder Collection-Referenz erstellt das System eine lokale Variable und initialisiert diese mit einem Link zu den Daten, die vom aufrufenden Programm geliefert werden. Der Parameter-Container im Unterprogramm entspricht dem der gelieferten Variablen: ein ein-, zwei- oder dreidimensionales Array oder eine Collection. Alle im Unterprogramm vorgenommenen Änderungen am Wert des Parameters wirken sich direkt auf den entsprechenden Wert der Daten des Aufrufprogramm aus.

Für eindimensionale Arrays ist es durch Angabe des ersten zugänglichen Elements im Array möglich, nur einen Teil des Arrays an das Unterprogramm zu übertragen. Für zwei- und dreidimensionale Arrays sowie Collections ist es nicht möglich, nur einen Teil des Arrays oder der Collection an das Unterprogramm zu übertragen. Die Variable muss dann ohne eckige Klammern [] und ohne Angabe eines Index oder Schlüssels übergeben werden. **VAL 3**-Konstantendaten oder das Ergebnis eines **VAL 3**-Ausdrucks können nicht per Array- oder Collection-Referenz übertragen werden.

Beispiel

`send1dMessage(string& s1dArray[])` ist ein Programm mit einem per Array-Referenz (eindimensional) übertragenen Einzelparameter.

`send2dMessage(string& s2dArray[])` ist ein Programm mit einem per Array-Referenz (zweidimensional) übertragenen Einzelparameter.

`send3dMessage(string& s3dArray[])` ist ein Programm mit einem per Array-Referenz (dreidimensional) übertragenen Einzelparameter.

`sendCollMessage(string& sMessageColl[""])` ist ein Programm mit einem per Collectionreferenz übertragenen Einzelparameter.

Keines dieser Programme kann mit einem konstanten Datenelement oder dem Ergebnis einer Berechnung genutzt werden:

```
// compilation errors: array variable expected as parameter
call send1dMessage("Waiting for signal StartCycle")
call send1dMessage("Waiting for signal"+l_sSignalName)
```

Der Container der übergebenen Variable muss mit dem für den Parameter angegebenen Container übereinstimmen:

```
// compilation errors: 1d array variable expected
call send1dMessage(sMessageColl)
call send1dMessage(s2dArray[12,3])
// compilation error: collection variable expected
call sendCollMessage(sMessage)
```

Mit Ausnahme von eindimensionale-Arrays ist es nicht möglich Arrays oder Collections teilweise zu übertragen. Arrays oder Collections müssen ohne Index oder Schlüssel angegeben werden.

```
// correct parameter
call send2dMessage(s2dArray)
call send3dMessage(s3dArray)
call sendCollMessage(sMessageColl)
call send1dMessage(sMessageArray)
call send1dMessage(sMessageArray[23])
// compilation errors: unexpected indices for the array
call send2dMessage(s2dArray[12,3])
call send3dMessage(s3dArray[5,7,9])
// compilation error: unexpected indices for the collection
call sendCollMessage(sMessageColl[l_sMessageName])
```

In letzterem Fall, wird nur der mit Index 23 startende Teil des Array `sMessageArray` an das Programm `send1dMessage()` übertragen. Auf Werte von `sMessageArray` mit einem Index unter 23 kann über `send1dMessage()` nicht zugegriffen werden.

KAPITEL 3

EINFACHE TYPEN

3.1. TYP BOOL

3.1.1. DEFINITION

Variablen oder Konstanten des Typs bool können folgende Werte annehmen:

- **true**: wahr
- **false**: falsch

Der vorprogrammierte Wert für eine Variable des Typs **bool** ist immer **false**.

3.1.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|---|
| bool <bool& bVariable> = <bbool bCondition> | Weist den Wert von bCondition der Variable bVariable zu und überträgt den Wert von bCondition . |
| bool <bool bCondition1> or <bbool bCondition2> | Überträgt den Wert des logischen OR zwischen bCondition1 und bCondition2 . bCondition2 wird nur zur Bewertung herangezogen, wenn bCondition1 false ist. |
| bool <bool bCondition1> and <bbool bCondition2> | Überträgt den Wert des logischen AND für bCondition1 und bCondition2 . bCondition2 wird nur zur Bewertung herangezogen, wenn bCondition1 true ist. |
| bool <bool bCondition1> xor bool <bCondition2> | Überträgt den Wert des ausschließenden logischen OR für bCondition1 und bCondition2 . |
| bool <bool bCondition1> != <bbool bCondition2> | Testet die Ungleichheit der Werte für bCondition1 und bCondition2 . Überträgt true , wenn die Werte verschieden sind, andernfalls false . |
| bool <bool bCondition1> == <bbool bCondition2> | Testet die Ungleichheit der Werte für bCondition1 und bCondition2 . Überträgt true , wenn die Werte identisch sind, andernfalls false . |
| bool ! <bbool bCondition> | Überträgt das Gegenteil des Wertes für bCondition |

Um Verwechslungen zwischen den Operatoren = und == zu vermeiden, ist der Operator = für als Anweisungsparameter verwendete VAL 3-Ausdrücke nicht zulässig. **if(bCondition1=bCondition2)** wird interpretiert als **bCondition1=bCondition2; if(bCondition1==true)**. Aber oft sollte Folgendes geschrieben werden: **if(bCondition1==bCondition2)**, Ein echter Unterschied!

3.2. TYP NUM

3.2.1. DEFINITION

Der Typ **num** stellt einen numerischen Wert mit etwa **14** signifikanten Stellen dar.

Numerische Berechnungen erfolgen daher mit einer durch die **14** Stellen beschränkten Genauigkeit.

Dies ist bei der Prüfung der Gleichheit zweier numerischer Werte zu berücksichtigen: Deshalb muss in der Regel in einem Intervall geprüft werden.

Konstanten numerischen Typs haben folgendes Format:

`[-] <digits>[.<digits>][e[-]<digits>]`

Das 'e' dient als Marker zur numerisch-wissenschaftlichen Bezeichnung anstelle von '10^A: 1e3 entspricht 1×10^3 (oder 1000), 1e-2 entspricht 1×10^{-2} (oder 0.01).

Variablen des Typs **num** werden vorprogrammiert auf den Wert **0** initialisiert.

Beispiel

Ein Test des Ergebnisses numerischer Berechnung muss die numerische Ungenauigkeit von Berechnungen berücksichtigen.

`if cos(nAngle)==0,if abs(cos(nAngle))<1e-10` should better be replaced with `if abs(cos(nAngle))<1e-10.`

Hier einige Konstantenzahlen:

```
1
0.2
-3.141592653
6.02214179e23
1.054571628e-34
```

3.2.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|---|
| num <num& nVariable> = <num nValue> | Weist nValue der Variable nVariable zu und überträgt den nValue . |
| bool <num nValue1> != <num nValue2> | Überträgt true , wenn nValue1 ungleich nValue2 ist, andernfalls false . |
| bool <num nValue1> == <num nValue2> | Überträgt true , wenn nValue1 gleich nValue2 ist, andernfalls false . |
| bool <num nValue1> >= <num nValue2> | Überträgt true , wenn nValue1 größer gleich nValue2 ist, andernfalls false . |
| bool <num nValue1> > <num nValue2> | Überträgt true , wenn nValue1 größer nValue2 ist, andernfalls false . |
| bool <num nValue1> <= <num nValue2> | Überträgt true , wenn nValue1 kleiner gleich nValue2 ist, andernfalls false . |
| bool <num nValue1> < <num nValue2> | Überträgt true , wenn nValue1 kleiner nValue2 ist, andernfalls false . |
| num <num nValue1> - <num nValue2> | Überträgt den Unterschied zwischen nValue1 und nValue2 . |
| num <num nValue1> + <num nValue2> | Überträgt die Summe von nValue1 und nValue2 . |
| num <num nValue1> % <num nValue2> | Modulo Operation: Überträgt den Rest der Division von nValue1 durch nValue2 . Ist nValue2 gleich 0 , wird ein Laufzeitfehler generiert. Das Zeichen für den Rest ist das Zeichen von nValue1 . |
| num <num nValue1> / <num nValue2> | Überträgt den Quotienten von nValue1 durch nValue2 . Ist nValue2 gleich 0 , wird ein Laufzeitfehler generiert. |
| num <num nValue1> * <num nValue2> | Überträgt das Produkt von nValue1 und nValue2 . |
| num - <num nValue> | Überträgt das Gegenteil von nValue . |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig. **nCos=cos(nAngle=30)** muss ersetzt werden durch **nAngle=30; nCos=cos(nAngle)**.

3.2.3. ANWEISUNGEN

num **sin**(num nAngle)

Funktion

Liefert den Sinus von **nAngle**.

Parameter

| | |
|-------------------|----------------|
| num nAngle | Winkel in Grad |
|-------------------|----------------|

Beispiel

`sin(30)` returns 0.5

num **asin**(num nValue)

Funktion

Liefert die Inverse des Sinus von **nValue** in Grad. Das Ergebnis liegt zwischen **-90** und **+90** Grad.

Ist **nValue** größer als **1** oder kleiner als **-1**, wird ein Laufzeitfehler generiert.

Beispiel

`asin(0.5)` returns 30

num **cos**(num nAngle)

Funktion

Liefert den Cosinus von **Angle**.

Parameter

| | |
|-------------------|----------------|
| num nAngle | Winkel in Grad |
|-------------------|----------------|

Beispiel

`cos(60)` returns 0.5

num **acos**(num nValue)

Funktion

Liefert die Inverse des Cosinus von **nValue** in Grad. Das Ergebnis liegt zwischen **0** und **180** Grad.

Ist **nValue** größer als **1** oder kleiner als **-1**, wird ein Laufzeitfehler generiert.

Beispiel

`acos(0.5)` returns 60

num **tan**(num nAngle)

Funktion

Liefert den Tangens von **Angle**.

Parameter

num nAngle Winkel in Grad

Beispiel

`tan(45)` returns 1.0

num atan(num nValue)

Funktion

Liefert die Inverse des Tangens von **nValue** in Grad. Das Ergebnis liegt zwischen **-90** und **+90** Grad.

Beispiel

`atan(1)` returns 45

num abs(num nValue)

Funktion

Liefert den Betrag von **nValue**.

Beispiel

Ein Test des Ergebnisses numerischer Berechnung muss die numerische Ungenauigkeit von Berechnungen berücksichtigen:

`if cos(nAngle)==0` should better be replaced with `if abs(cos(nAngle))<1e-10`.

`abs(3.1415)` returns 3.1415
`abs(-3.1415)` returns 3.1415

num sqrt(num nValue)

Funktion

Liefert die Quadratwurzel von **nValue**.

Ist **nValue** negativ, wird ein Laufzeitfehler generiert.

Beispiel

`sqrt(9)` returns 3

num exp(num nValue)

Funktion

Liefert den Exponentialwert von **nValue**.

Ist **nValue** zu groß, wird ein Laufzeitfehler erzeugt.

Beispiel

`exp(1)` returns 2.718281828459

num power(num nX, num nY)

Funktion

Liefert **nX** hoch **nY**: nX^{nY}

Ist **nX** negativ oder null oder das Ergebnis zu groß, wird ein Laufzeitfehler generiert.

Beispiel

Dieses Programm berechnet auf 2 verschiedene Arten 5 hoch 7.

```
// First way: power instruction
nResult = power(5,7)
// Second way: power(x,y)=exp(y*ln(x)) (with numerical inaccuracy)
nResult = exp(7*ln(5))
```

num ln(num nValue)

Funktion

Liefert den natürlichen Logarithmus von **nValue**.

Ist **nValue** negativ oder null, wird ein Laufzeitfehler generiert.

Beispiel

```
ln(2.718281828) returns 0.99999999983113
```

num log(num nValue)

Funktion

Liefert den Zehnerlogarithmus von **nValue**.

Ist **nValue** negativ oder null, wird ein Laufzeitfehler generiert.

Beispiel

```
log(1000) returns 3
```

num roundUp(num nValue)

Funktion

Liefert den nächstgrößeren ganzzahligen **nValue**.

Beispiel

```
roundUp(7.8) returns 8
roundUp(-7.8) returns -7
```

num roundDown(num nValue)

Funktion

Liefert den nächstkleineren ganzzahligen **nValue**.

Beispiel

```
roundDown(7.8) returns 7
roundDown(-7.8) returns -8
```

num round(num nValue)

Funktion

Liefert den nächstliegenden ganzzahligen **nValue**.

Beispiel

```
round(7.8) returns 8
round(-7.8) returns -8
round(0.5) returns 1
round(-0.5) returns 0
```

num min(num nX, num nY)

Funktion

Liefert den kleineren Wert von **nX** und **nY**.

Beispiel

```
min(-1,10) returns -1
```

num max(num nX, num nY)

Funktion

Liefert den größeren Wert von **nX** und **nY**.

Beispiel

```
max(-1,10) returns 10
```

num limit(num nValue, num nMin, num nMax)

Funktion

Liefert den durch die Werte **nMin** und **nMax** begrenzten **nValue**.

Beispiel

```
limit(30,-90,90) returns 30
limit(100,-90,90) returns 90
limit(-100,-90,90) returns -90
```

num sel(bool bCondition, num nValue1, num nValue2)

Funktion

Liefert **nValue1**, wenn **bCondition** gleich **true** ist, ansonsten **nValue2**.

Beispiel

```
sel(true,-90, 90) returns -90
sel(false,-90, 90) returns 90
```

3.3. BITFELD-TYP

3.3.1. DEFINITION

Ein Bitfeld ist eine Möglichkeit zur kompakten Speicherung und Austausch einer Bitfolge (Boolesche Werte oder digitale Eingänge/Ausgänge). **VAL 3** liefert keinen spezifischen Datentyp zur Handhabung von Bitfeldern, nutzt aber den num-Typ, um ein 32-Bitfeld als einen positiven ganzzahligen Wert im Bereich [0, 2^{32}] zu speichern.

Alle numerischen **VAL 3**-Werte können als 32-Bit Bitfelder gesehen werden; die Anweisungen zur Bitfeld-Handhabung sehen automatisches Aufrunden eines numerischen Werts auf einen 32-Bit positiven ganzzahligen Wert vor, der dann als 32-Bit Bitfeld behandelt wird.

3.3.2. OPERATOREN

Die standardmäßigen Operatoren des num-Typs gelten für ein Bitfeld: '=', '==', '!=',

3.3.3. ANWEISUNGEN

num bNot(num nBitField)

Funktion

Die Anweisung überträgt die bitweise logische Operation 'nicht' auf ein 32-Bit Bitfeld. (Das i-te Bit des Ergebnisses wird auf 1 gesetzt, wenn das i-te Bit der Eingabe 0 beträgt). Deshalb ist dieses Ergebnis ein positiver ganzzahliger Wert im Bereich [0, 2^{32}].

Die numerische Eingabe wird vor der bitweisen Operation auf einen positiven ganzzahligen Wert im Bereich [0, 2^{32}] aufgerundet.

Beispiel

Dieses Programm setzt die Bits i bis j eines Bitfelds nBitField mithilfe der Maske nMask zurück.

```
// Compute a bit mask with bits i to j set to 1 (see bOr for explanations)
nMask=(power(2,j-i+1)-1)*power(2,i)
// Invert the mask to have all bits to 1 except bit i to j
nMask=bNot(nMask)
// Reset bits i to j using the bitwise 'and'
nBitField=bAnd(nBitField, nMask)
```

num bAnd(num nBitField1, num nBitField2)

Funktion

Die Anweisung überträgt die bitweise logische Operation 'und' auf zwei 32-Bit Bitfelder. (Das i-te Bit des Ergebnisses wird auf 1 gesetzt, wenn die i-ten Bits beider Eingaben auf 1 gesetzt sind). Deshalb ist dieses Ergebnis ein positiver ganzzahliger Wert im Bereich [0, 2^{32}].

Die numerischen Eingaben werden vor der bitweisen Operation auf einen positiven ganzzahligen Wert im Bereich [0, 2^{32}] aufgerundet.

Beispiel

Dieses Programm zeigt ein 32-Bit Bitfeld nBitField auf dem Bildschirm an, indem jedes Bit hintereinander getestet wird:

```
for i=31 to 0 step -1
  // Compute the mask for the i th bit
  nMask=power(2,i)
  if bAnd(nBitField, nMask)==nMask
    sOutput="1"
  else
    sOutput="0"
  endIf
endFor
```

num bOr(num nBitField1, num nBitField2)

Funktion

Die Anweisung überträgt die bitweise logische Operation 'oder' auf zwei 32-Bit Bitfelder. (Das i-te Bit des Ergebnisses wird auf 1 gesetzt, wenn das i-te Bit mindestens eines Parameters auf 1 gesetzt ist). Deshalb ist dieses Ergebnis ein positiver ganzzahliger Wert im Bereich [0, 2^{32}].

Die numerischen Eingaben werden vor der bitweisen Operation auf einen positiven ganzzahligen Wert im Bereich [0, 2^{32}] aufgerundet.

Beispiel

Dieses Programm berechnet auf zwei verschiedene Arten eine Bitfeld-Maske, wobei die Bits i bis j gesetzt werden.

```
// First way: logical 'or' on bits i to j
nBitField=0
for k=i to j
  nBitField=bOr(nBitField, power(2,k))
endFor
// Second way: compute a bit mask of (j-i) bits
nBitField=(power(2,j-i+1)-1)
// Then shift the bit mask by i bits
nBitField=nBitField*power(2,i)
```

num bXor(num nBitField1, num nBitField2)

Funktion

Die Anweisung überträgt die bitweise logische Operation 'xor' (ausschließliches oder) auf zwei 32-Bit Bitfelder. (Das i-te Bit des Ergebnisses wird auf 1 gesetzt, wenn die i-ten Bits der beiden Parameter verschieden sind). Deshalb ist dieses Ergebnis ein positiver ganzzahliger Wert im Bereich [0, 2^{32}].

Die numerischen Eingaben werden vor der bitweisen Operation auf einen positiven ganzzahligen Wert im Bereich [0, 2^{32}] aufgerundet.

Beispiel

Dieses Programm kehrt auf dem Bitfeld nBitField i-Bits auf j um:

```
// Compute mask for bits i to j (see bOr example)
nMask=(power(2,j-i+1)-1)*power(2,i)
// Invert bits i to j using the mask
nBitField=bXor(nBitField,nMask)
```

**num toBinary(num nValue[], num nValueSize, String sDataFormat,
num& nDataByte[])**

**num fromBinary(num nDataByte[], num nDataSize,
String sDataFormat, num& nValue[])**

Funktion

Die Anweisungen **toBinary/fromBinary** sollen den Austausch numerischer Werte zwischen zwei Geräten mithilfe einer seriellen Schnittstelle oder einer Netzwerkverbindung ermöglichen. Die numerischen Werte werden zuerst in einen Byte-Strom umgewandelt. Dann werden die Bytes an die Gegenstelle gesendet. Schließlich wandelt die Gegenstelle die Bytes wieder um, um die ursprünglichen numerischen Werte wiederherzustellen. Verschiedene binäre Umwandlungen numerischer Werte sind möglich.

Die Anweisung **toBinary** wandelt numerische Werte in ein Byte-Array (8-Bit Bitfeld, positiver ganzzahliger Wert im Bereich [0, 255]) wie durch das Datenformat **sDataFormat** angegeben. Die Anzahl der umzuwandelnden numerischen Werte **nValue** wird vom Parameter **nValueSize** angegeben. Das Ergebnis wird im Array **nDataByte** gespeichert und die Anweisung überträgt die Anzahl umgewandelter Bytes in diesem Array.

Wenn die Anzahl zu verschlüsselnder Werte **nValueSize** größer ist als die Größe von **nValue**, wenn das spezifizierte Format nicht unterstützt wird oder das Ergebnis-Array **nDataByte** nicht groß genug ist, um alle Eingabedaten aufzunehmen, wird ein Laufzeitfehler erzeugt.

Die Anweisung **fromBinary** wandelt ein Byte-Array in numerische Werte **nValue**, wie vom Datenformat **sDataFormat** angegeben. Die Anzahl der umzuwandelnden Bytes wird vom Parameter **nDataSize** angegeben. Das Ergebnis wird im Array **nValue** gespeichert und die Anweisung überträgt die Anzahl an Werten in diesem Array. Wenn binäre Daten beschädigt sind (Bytes außerhalb des Bereichs [0, 255] oder ungültige Gleitkomma-Umwandlung) überträgt die Anweisung das Gegenteil der Anzahl korrekt umgewandelter Werte (negativer Wert).

Wenn die Anzahl zu verschlüsselnder Bytes **nDataSize** größer ist als die Größe von **nDataByte**, wenn das spezifizierte Format nicht unterstützt wird oder das Ergebnis-Array **nValue** nicht groß genug ist, um alle Eingabedaten aufzunehmen wird ein Laufzeitfehler erzeugt.

Die unterstützten binären Formate finden sich in unten stehender Tabelle:

- Das Zeichen "-" gibt die Umwandlung einer vorzeichenbehafteten ganzen Zahl an (das letzte Bit des Bitfelds verschlüsselt das Vorzeichen des Werts).
- Die Ziffer gibt die Anzahl Bytes für die Umwandlung jedes numerischen Wertes an.
- Die Erweiterung ".0" kennzeichnet die Umwandlung von Gleitkommawerten (es werden Umwandlungen von IEEE 754 32-Bit und 64-Bit Gleitkommazahlen unterstützt).
- Der letzte Buchstabe gibt die Byte-Reihenfolge an: "l" steht für 'Little-Endian' (das Byte mit der geringsten Wertigkeit wird zuerst angegeben), "b" steht für 'Big-Endian' (das höherwertige Byte wird zuerst angegeben). Das 'big endian'-Format ist der Standard für Netzwerkanwendungen (TCP/IP).

| | |
|--------|---|
| "-1" | Byte mit Vorzeichen |
| "1" | Byte ohne Vorzeichen |
| "-2l" | Wort mit Vorzeichen, Little-Endian |
| "-2b" | Wort mit Vorzeichen, Big-Endian |
| "2l" | Wort ohne Vorzeichen, Little-Endian |
| "2b" | Wort ohne Vorzeichen, Big-Endian |
| "-4l" | Doppelwort mit Vorzeichen, Little-Endian |
| "-4b" | Doppelwort mit Vorzeichen, Big-Endian |
| "4l" | Doppelwort ohne Vorzeichen, Little-Endian |
| "4b" | Doppelwort ohne Vorzeichen, Big-Endian |
| "4.0l" | Gleitkommawert mit einfacher Genauigkeit, Little-Endian |
| "4.0b" | Gleitkommawert mit einfacher Genauigkeit, Big-Endian |
| "8.0l" | Gleitkommawert mit doppelter Genauigkeit, Little-Endian |
| "8.0b" | Gleitkommawert mit doppelter Genauigkeit, Big-Endian |

Das **VAL 3**-Format für numerische Variablen ist eine Gleitkommazahl mit doppelter Genauigkeit. Dieses Format muss benutzt werden, um numerische Werte ohne Genauigkeitsverlust auszutauschen.

Beispiel

Das erste Programm wandelt die **trsf**-Variable **trShiftOut** in das Byte-Array **nByteOut** und sendet sie über die Verbindung **siTcpClient**. Das zweite Programm liest die Bytes von der Verbindung **siTcpServer** und konvertiert sie zurück in die Variable **trsf trShiftIn**.

```
// ---- Program to send a trsf ----
// Copy the trsf coordinates into a numerical buffer
nTrsfOut[0]=trShiftOut.x
nTrsfOut[1]=trShiftOut.y
nTrsfOut[2]=trShiftOut.z
nTrsfOut[3]=trShiftOut.rx
nTrsfOut[4]=trShiftOut.ry
nTrsfOut[5]=trShiftOut.rz
// Encode 6 numerical values (double precision floating point, therefore 8 bytes) into
6*8=48 bytes in nByteOut[48] array
toBinary(nTrsfOut, 6, "8.0b", nByteOut)
// Send nByte array (48 bytes) through tcpClient
sioSet(siTcpClient, nByteOut)

// ---- Program to read a trsf ----
nb=0
i=0
while (nb<48)
  nb=sioGet(siTcpServer, nByteIn[i])
  if(nb>0)
    i=i+nb
  else
    // Communication error
    return
  endIf
endWhile
if (fromBinary(nByteIn, 48, "8.0b", nTrsfIn) != 6)
  // Corrupted data
  return
else
  trShiftIn.x=nTrsfIn[0]
  trShiftIn.y=nTrsfIn[1]
  trShiftIn.z=nTrsfIn[2]
  trShiftIn.rx=nTrsfIn[3]
  trShiftIn.ry=nTrsfIn[4]
  trShiftIn.rz=nTrsfIn[5]
endif
```

3.4. TYP STRING

3.4.1. DEFINITION

Variablen des Typs Zeichenkette (string) dienen zur Speicherung von Texten. Der Typ string unterstützt den standardmäßigen Unicode-Zeichensatz. Beachten Sie, dass die korrekte Anzeige eines Unicode-Zeichens von den am Anzeigegerät installierten Zeichensatztabellen abhängt.

Eine Zeichenkette ist maximal 128 Byte groß; die maximale Zeichenanzahl in einer Zeichenkette hängt davon ab, welche Zeichen verwendet werden, da die interne Zeichencodierung (Unicode UTF8) zwischen 1 Byte (für ASCII-Zeichen) und 4 Byte (3 für chinesische Schriftzeichen) benutzt.

Die maximale Länge einer ASCII-Zeichenkette beträgt daher 128 Zeichen; die maximale Länge einer Zeichenkette chinesischer Schriftzeichen beträgt 42 Zeichen.

Variablen des Typs String werden vorprogrammiert auf den Wert "" (leerer String) initialisiert.

3.4.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|---|
| string <string& sVariable> = <string sString> | Weist sString der Variable sVariable zu und überträgt den sString . |
| bool <string sString1> != <string sString2> | Liefert true , wenn sString1 und sString2 nicht identisch sind, ansonsten false . |
| bool <string sString1> == <string sString2> | Liefert true , wenn sString1 und sString2 identisch sind, ansonsten false . |
| string <string sString1> + <string <sString2> | Liefert die ersten Zeichen (begrenzt auf 128 Byte) von sString1 verknüpft mit sString2 . |

Um Verwechslungen zwischen den Operatoren = und == zu vermeiden, ist der Operator = für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig. **nLen=len(sString="hello wild world")** muss ersetzt werden durch **sString="hello wild world"; nLen=len(sString)**.

3.4.3. ANWEISUNGEN

string toString(string sFormat, num nValue)

Funktion

Diese Anweisung liefert eine Zeichenketten die **nValue** im Anzeigeformat **sFormat** darstellt.

Das Format hat die Form "**size.precision**", wobei **size** für die Mindestgröße des Ergebnisses (gegebenenfalls werden am Anfang der Zeichenkette Leerstellen hinzugefügt) und **precision** für die Anzahl bedeutungstragender Ziffern nach dem Dezimalkomma steht (die **0** am Ende der Kette werden durch Leerstellen ersetzt). **size** und **precision** haben den Standardwert **0**. Der ganzzahlige Teil des Werts wird niemals verkürzt, auch wenn seine Anzeigelänge den Wert von **size** übertrifft.

Beispiel

überträgt

```
nPi = 3.141592654
toString(.4, nPi) returns "3.1416"
toString(8, nPi) returns "      3" (7 spaces before the '3'
toString(8.4, nPi) returns " 3.1416" (2 spaces before the '3')
toString(8.4, 2.70001) returns " 2.7  " (2 spaces before the '2', 3 spaces after the
'7')
toString("", nPi) returns "3"
toString("1.2", 1234.1234) returns "1234.12"
```

Siehe auch

string chr(num nCodePoint)
string toNum(string sString, num& nValue, bool& bReport)

string toNum(string sString, num& nValue, bool& bReport)

Funktion

Diese Anweisungen finden den numerischen **nValue** am Anfang der genannten **sString**, und liefern **sString**, in der alle Zeichen bis zum nächsten numerischen **value** entfernt worden sind.

Befindet sich am Anfang von **sString** kein numerischer Wert, so wird **bReport** auf **false** gesetzt und **nValue** wird nicht geändert, andernfalls wird **bReport** auf **true** gesetzt.

Beispiel

```
toNum("10 20 30", nVal, bOk) returns "20 30", nVal equals 10, bOk equals true
toNum("a10 20 30", nVal, bOk) returns "a10 20 30", nVal is unchanged, bOk equals false
toNum("10 end", nVal, bOk) returns "", nVal equals 10, bOk equals true
```

This program displays successively 90, 0, -7.6, 17.3

```
sBuffer = "+90 0.0 -7.6 17.3"
do
    sBuffer = toNum(sBuffer, nVal, bOk)
    sOutput = toString("", nVal)
until (sBuffer=="") or (bOk != true)
```

Siehe auch

string toString(string sFormat, num nValue)

string **chr(num nCodePoint)**

Funktion

Diese Anweisung liefert das entsprechende Unicode-Zeichen, wenn es eine gültige Unicode-Nummer ist. Andernfalls liefert es eine leere Zeichenkette.

In der nachstehenden Tabelle sind die Unicode-Nummern zusammengestellt, die kleiner als **128** sind (entspricht der **ASCII**-Zeichentabelle). Die grau dargestellten Zeichen sind Kontrollcodes, die durch ein Fragezeichen ersetzt werden können, wenn die Zeichenkette angezeigt wird.

Alle gültigen Unicode-Nummern werden vom **VAL 3** String-Typ unterstützt. Die korrekte Anzeige der Zeichen hängt jedoch von den am Anzeigegerät installierten Zeichensätzen ab. Die vollständige Liste der Unicode-Zeichen finden Sie unter <http://www.unicode.org> (nach 'Code Charts' suchen).

| | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| NUL | SOH | STX | ETX | EOT | ENQ | ACQ | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| " " | ! | " | #" | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
| @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 |
| ' | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |
| p | q | r | s | t | u | v | w | x | y | z | (|) |) | ~ | DEL |

Beispiel

`chr(65)` returns "A"

Siehe auch

`num asc(string sText, num nPosition)`

num **asc(string sText, num nPosition)**

Funktion

Diese Anweisung liefert die Unicode-Nummer des Zeichens an der Stelle **nPosition**.

Wird -1, wenn **nPosition** negativ oder größer als die spezifizierte Textlänge ist.

Beispiel

```
asc("A", 0) returns 65
```

Siehe auch

string chr(num nCodePoint)

string **left(string sText, num nSize)**

Funktion

Diese Anweisungen liefern die **nSize** ersten Zeichen von **sText**. Wenn **nSize** größer ist als die Länge von **sText**, liefert diese Anweisung die **sText**.

Ist **nSize** negativ, wird ein Laufzeitfehler generiert.

Beispiel

```
left("hello world", 5) returns "hello"
left("hello world", 20) returns "hello world"
```

string **right(string sText, num nSize)**

Funktion

Diese Anweisung liefert die **nSize** letzten Zeichen von **sText**. Wenn die spezifizierte Anzahl größer ist, als die Länge von **sText**, liefert diese Anweisung **sText**.

Ist **nSize** negativ, wird ein Laufzeitfehler generiert.

Beispiel

```
right("hello world", 5) returns "world"
right("hello world", 20) returns "hello world"
```

string **mid(string sText, num nSize, num nPosition)**

Funktion

Liefert **nSize** Zeichen von **sText** ab dem Zeichen **nPosition** bis zum Ende von **sText**.

Sind **nPosition** oder **sText** negativ, wird ein Laufzeitfehler generiert.

Beispiel

```
mid("hello wild world", 4, 6) returns "wild"
mid("hello wild world", 20, 6) returns "wild world"
```

string insert(string sText, string sInsertion, num nPosition)

Funktion

Diese Anweisung überträgt die **sText**, in die **sInsertion** nach dem Zeichen an der Stelle **nPosition** eingefügt wurde. Ist **nPosition** größer als die Größe von **sText**, so wird **sInsertion** am Ende von **sText** angefügt. Das Ergebnis wird abgeschnitten, wenn es größer als 128 Byte ist.

Ist **nPosition** negativ, wird ein Laufzeitfehler generiert.

Beispiel

```
insert("hello world", "wild", 6) returns "hello wild world"
```

string delete(string sText, num nSize, num nPosition)

Funktion

Diese Anweisung überträgt die **sText**, in der die **nSize** Zeichen nach dem Zeichen an der Stelle **nPosition** gelöscht wurden. Wenn **nPosition** größer ist als die Länge von **sText**, überträgt die Anweisung **sText**.

Sind **nSize** oder **nPosition** negativ, wird ein Laufzeitfehler generiert.

Beispiel

```
delete("hello wild world", 5, 6) returns "hello world"
```

string replace(string sText, string sReplacement, num nSize, num nPosition)

Funktion

Diese Anweisung überträgt **sText**, in der die **nSize** Zeichen nach dem Zeichen an der Stelle **nPosition** durch **sReplacement** ersetzt wurden. Wenn **nPosition** größer ist als die Länge von **sText**, überträgt die Anweisung **sText**.

Sind **nSize** oder **nPosition** negativ, wird ein Laufzeitfehler generiert.

Beispiel

```
replace("hello ? world", "wild", 1, 6) returns "hello wild world"
```

num find(string sText1, string sText2)

Funktion

Diese Anweisung überträgt die Stelle (zwischen **0** und **127**) des ersten Zeichens beim erstmaligen Auftreten von **sText2** in **sText1**. Wenn **sText2** nicht in **sText1** enthalten ist, liefert die Anweisung **-1**.

Beispiel

```
find("hello wild world", "wild") returns 6
```

num len(string sText)

Funktion

Diese Anweisung liefert die Anzahl an Zeichen in **sText**.

Beispiel

```
len("hello wild world") returns 16
```

Siehe auch

num getDisplayLen(string sText)

3.5. TYP DIO

3.5.1. DEFINITION

Variablen des Typ **dio** werden zur Verknüpfung einer **VAL 3**-Applikation mit digitalen Ein-/Ausgängen des Systems benutzt. Eine **dio**-Variable speichert einen Link zu einem digitalen Ein-/Ausgang des Systems, der "physischen Adresse".

Anweisungen, in denen eine Variable des Typs **dio** verwendet wird, die nicht einem im System deklarierten Ein-/Ausgang zugewiesen wurde, bewirken einen Laufzeitfehler. Variablen des Typs **dio** werden vorprogrammiert auf einen undefinierten Link initialisiert. Der Link einer **dio**-Variablen kann von einer anderen **dio**-Variablen aus initialisiert werden, vom **MCP** aus oder im **VAL 3 Studio** aus der **Stäubli Robotics Suite**.

3.5.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|--|
| bool <dio diOutput> = <bool bCondition> | Weist bCondition dem Status des diOutput zu und überträgt bCondition . Wenn diOutput keinem Ausgang des Systems zugewiesen ist, wird ein Laufzeitfehler erzeugt. |
| bool <dio dilnput1> != <bool bInput2> | Überträgt true , wenn dilnput1 und bInput2 nicht den gleichen Status haben, ansonsten false . |
| bool <dio dilnput> != <bool bCondition> | Überträgt true , wenn der Status des dilnput nicht gleich bCondition ist, ansonsten false . |
| bool <dio dilnput> == <bool bCondition> | Überträgt true , wenn der Status des dilnput gleich bCondition ist, ansonsten false . |
| bool <dio dilnput1> == <dio dilnput2> | Überträgt true , wenn dilnput1 und dilnput2 den gleichen Status haben, ansonsten false . |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig. **if(diOutput=dilnput)** wird interpretiert als: **diOutput=dilnput; if(diOutput==true)**. Aber oft sollte Folgendes geschrieben werden: **if(diOutput==dilnput)**, Ein echter Unterschied!

ACHTUNG:

Der Operator **'='** zwischen zwei **dio** Variablen existiert nicht mehr in **VAL 3 s7**, um die Konsistenz mit anderen **'='** Operatoren zu gewährleisten (siehe die Definition des **'='** Operators für Benutzertypen). Er kann einfach durch den **'='** Operator zwischen einem **dio** und einem **bool** ersetzt werden: **diOut = diln** früherer **VAL 3** Versionen kann durch **diOut = (diln==true)** ersetzt werden.

3.5.3. ANWEISUNGEN

void dioLink(dio& diVariable, dio diSource)

Funktion

Diese Anweisung weist **diVariable** dem Ein-/Ausgang des Systems zu, an dem **diSource** angeschlossen ist.

Beispiel

Diese Applikation nutzt ein Signal, das mit verschiedenen Hardwaregeräten konfiguriert werden kann. Das unten stehende Programm testet welches Gerät installiert ist, um die **diSignal**-Variable zu initialisieren, die dann im Rest der Applikation verwendet wird.

```
if(ioStatus(diDevice1Signal)>=0)
    // device 1 is installed: use it
    dioLink(diSignal, diDevice1Signal)
elseIf (ioStatus(diDevice2Signal)>=0)
    // device 2 is installed: use it
    dioLink(diSignal, diDevice2Signal)
else
    sOutput="Error: no io device installed"
endif
```

num dioGet(dio diArray[])

Funktion

Diese Anweisung liefert den Zahlenwert von **diArray**, welcher wie eine ganze Zahl in Binärschreibweise gelesen wird, das heißt: **diArray[0] + 2*diArray[1] + 4*diArray[2] + ... + 2^k*diArray[k]**, oder **diArray[i] = 1** wenn **diArray[i]** gleich **true** ist, anderenfalls 0.

Ein Laufzeitfehler wird erzeugt, wenn ein Element von **diArray** nicht an einem Ein-/Ausgang des Systems angeschlossen ist.

Beispiel

```
diArray[0] = false
diArray[1] = true
diArray[2] = false
diArray[3] = true
dioGet(diArray) returns 10 = 0+2*1+4*0+8*1
```

Siehe auch

num dioSet(dio diArray[], num nValue)

num dioSet(dio diArray[], num nValue)

Funktion

Diese Anweisung konvertiert den ganzzahligen Teil von **nValue** in Binärdarstellung, weist ihn den Ausgängen von **diArray** zu und gibt den entsprechenden Wert zurück, zum Beispiel:

diArray[0] + 2 * diArray[1] + 4 * diArray[2] + ... + 2^k * diArray[k], oder **diArray[i] = 1** wenn **diArray[i]** gleich **true** ist, anderenfalls **0**.

Ist ein Element der **diArray** nicht an einem Ausgang des Systems angeschlossen, so wird ein Laufzeitfehler erzeugt.

Beispiel

Benutzen von **di4bitsArray**, Array der Größe 4:

```
dioSet(di4bitsArray, 10) returns 10
dioSet(di4bitsArray, 26) returns 10, because 26 requires 5 bits in a binary encoding: 10 = 26 - 2^4
```

Siehe auch

num dioGet(dio diArray[])

num ioStatus(dio diInputOutput)

Funktion

Diese Anweisung liefert eine positive Zahl, wenn die angegebene Ein-/Ausgangsvariable funktioniert und eine negative Zahl, wenn ein Fehler vorliegt. Der zurückgegebene Wert detailliert den Status des Ein-/Ausgangs:

| | |
|----|--|
| 0 | Der Ein-/Ausgang funktioniert. |
| 1 | Der Ein-/Ausgang funktioniert, wurde aber vom Bediener verriegelt. Eingänge haben dann einen festen Wert (vom Bediener kontrolliert), der sich vom Hardware-Wert unterscheiden kann. Ausgänge haben dann einen festen Wert, der vom Bediener kontrolliert wird: Schreiben an den Ausgang hat keinerlei Wirkung. Der Verriegelungsmodus ist ein Mittel zur Fehlersuche. |
| 2 | Der Ein-/Ausgang wird simuliert (Software-Ein-/Ausgang, keine Auswirkung auf die Hardware). |
| -1 | Der Ein-/Ausgang funktioniert nicht, weil der Link (physische Adresse) nicht definiert ist. |
| -2 | Der Ein-/Ausgang funktioniert nicht, weil der Link (physische Adresse) keinem Ein-/Ausgangssystem entspricht. Das der physischen Adresse entsprechende Hardware-Gerät ist entweder nicht installiert oder konnte nicht initialisiert werden. |
| -3 | Der Ein-/Ausgang funktioniert nicht, weil ein Fehler am Ein-/Ausgangsgerät vorliegt. |

Beispiel

Diese Applikation nutzt ein Signal, das mit verschiedenen Hardwaregeräten konfiguriert werden kann. Das unten stehende Programm testet welches Gerät installiert ist, um die **diSignal**-Variable zu initialisieren, die dann im Rest der Applikation verwendet wird.

```
if(ioStatus(diDevice1Signal)>=0)
  // device 1 is installed: use it
  dioLink(diSignal, diDevice1Signal)
elseif (ioStatus(diDevice2Signal)>=0)
  // device 2 is installed: use it
  dioLink(diSignal, diDevice2Signal)
else
  sOutput="Error: no io device installed"
endif
```

Siehe auch

num ioStatus(dio dilnputOutput, string& sDescription, string& sPhysicalPath)
num ioStatus(aio ailnputOutput)

num ioStatus(dio dilnputOutput, string& sDescription, string& sPhysicalPath)

Funktion

Die Anweisung funktioniert genau wie die oben beschriebene ioStatus-Anweisung, liefert aber zusätzlich den Beschreibungstext und den Link (physische Adresse) für den angegebenen Ein-/Ausgang.

Die Beschreibung ist ein freier, mit den Werkzeugen zur Konfiguration der Ein-/Ausgänge definierter Text. Das Format des physischen Links ist vom Ein-/Ausgangsgerät abhängig. Üblicherweise hat es folgende Form: 'boardName\moduleName\ioAdress' oder eine einmalige ID-Zeichenkette wie 'C792E6DA-FEA0-44D7-BECF-FFB6CB1B6577'.

Beispiel

Dieses Programm testet ein Signal und zeigt Fehlermeldungen an, wenn es nicht funktioniert.

```
if ioStatus(diSignal, sDescription, sPath)<0  
    sOutput="Signal "+sPath+" in error "+Description:+sDescription  
endif
```

Siehe auch

num ioStatus(aio ailnputOutput)
num ioStatus(dio dilnputOutput)

3.6. TYP AIO

3.6.1. DEFINITION

Variablen des Typs **aio** werden zur Verknüpfung einer **VAL 3**-Applikation mit analogen Ein-/Ausgängen des Systems benutzt. Eine **aio**-Variable speichert einen Link zu einem analogen Ein-/Ausgang des Systems, der "physischen Adresse".

Anweisungen, in denen eine Variable des Typs **aio** verwendet wird, die nicht einem im System deklarierten Ein-/Ausgang zugewiesen wurde, bewirken einen Laufzeitfehler. Variablen des Typs **aio** werden vorprogrammiert auf einen undefinierten Link initialisiert. Der Link einer **aio**-Variable kann von einer anderen **aio**-Variable, vom Roboter **MCP** (nur SRC-Serienversion) oder durch Verwendung von **VAL 3 Studio** in **Stäubli Robotics Suite** aus initialisiert werden.

3.6.2. ANWEISUNGEN

void aioLink(aio& aiVariable, aio aiSource)

Funktion

Diese Anweisung weist **aiVariable** dem Ein-/Ausgang des Systems zu, an dem **aiSource** angeschlossen ist.

Beispiel

Diese Applikation nutzt ein Signal, das mit verschiedenen Hardwaregeräten konfiguriert werden kann. Das unten stehende Programm testet welches Gerät installiert ist, um die **aiSignal**-Variable zu initialisieren, die dann im Rest der Applikation verwendet wird.

```
if(ioStatus(aiDevice1Signal)>=0)
  // device 1 is installed: use it
  aioLink(aiSignal, aiDevice1Signal)
elseif (ioStatus(aiDevice2Signal)>=0)
  // device 2 is installed: use it
  aioLink(aiSignal, aiDevice2Signal)
else
  sOutput="Error: no io device installed"
endif
```

num aioGet(aio ailnput)

Funktion

Diese Anweisung liefert den numerischen Wert von **ailnput**.

Wenn **ailnput** keinem Ein-/Ausgang des Systems zugewiesen ist, wird ein Laufzeitfehler erzeugt.

Beispiel

aioGet(aiSensor) returns the current sensor value

Siehe auch

num aioSet(aio aiOutput, num nValue)

num aioSet(aio aiOutput, num nValue)

Funktion

Diese Anweisung weist **aiOutput** den Wert **nValue** zu und liefert **nValue** zurück. Wenn der eingestellte Wert sich außerhalb des aio-Bereichs befindet, entspricht der Rückgabewert dem aktuellen Wert des aio-Ausgangs.

Wenn **aiOutput** keinem Ausgang des Systems zugewiesen ist, wird ein Laufzeitfehler erzeugt.

Beispiel

`aioSet(aiCommand, -12.3)` übergibt -12.3 an den Ausgangsbefehl und liefert -12.3 falls aiCommand ein Gleitkomma-Ausgang ist.

`aioSet(aiCommand, 12.3)` übergibt 12 an den Ausgangsbefehl und liefert 12 falls aiCommand ein Ganzzahl-Ausgang ist.

Siehe auch

[num aioGet\(aio aiInput\)](#)

num ioStatus(aio ailnputOutput)

Funktion

Diese Anweisung liefert eine positive Zahl, wenn die angegebene Ein-/Ausgangsvariable funktioniert und eine negative Zahl, wenn ein Fehler vorliegt. Der zurückgegebene Wert detailliert den Status des Ein-/Ausgangs:

| | |
|----|--|
| 0 | Der Ein-/Ausgang funktioniert. |
| 1 | Der Ein-/Ausgang funktioniert, wurde aber vom Bediener verriegelt. Eingänge haben dann einen festen Wert (vom Bediener kontrolliert), der sich vom Hardware-Wert unterscheiden kann. Ausgänge haben dann einen festen Wert, der vom Bediener kontrolliert wird: Schreiben an den Ausgang hat keinerlei Wirkung. Der Verriegelungsmodus ist ein Mittel zur Fehlersuche. |
| 2 | Der Ein-/Ausgang wird simuliert (Software-Ein-/Ausgang, keine Auswirkung auf die Hardware). |
| -1 | Der Ein-/Ausgang funktioniert nicht, weil der Link (physische Adresse) nicht definiert ist. |
| -2 | Der Ein-/Ausgang funktioniert nicht, weil der Link (physische Adresse) keinem Ein-/Ausgangssystem entspricht. Das der physischen Adresse entsprechende Hardware-Gerät ist entweder nicht installiert oder konnte nicht initialisiert werden. |
| -3 | Der Ein-/Ausgang funktioniert nicht, weil ein Fehler am Ein-/Ausgangsgerät vorliegt. |

Beispiel

Diese Applikation nutzt ein Signal, das mit verschiedenen Hardwaregeräten konfiguriert werden kann. Das unten stehende Programm testet welches Gerät installiert ist, um die aiSignal-Variable zu initialisieren, die dann im Rest der Applikation verwendet wird.

```

if(ioStatus(aiDevice1Signal)>=0)
// Gerät 1 ist installiert: nutze es
aioLink(aiSignal, aiDevice1Signal)
elseif (ioStatus(aiDevice2Signal)>=0)
// Gerät 2 ist installiert: nutze es
aioLink(aiSignal, aiDevice2Signal)
else
sOutput="Error: no io device installed"
endif

```

Siehe auch

[num ioStatus\(dio dilnputOutput\)](#)

**num ioStatus(aio dilnputOutput, string& sDescription,
 string& sPhysicalPath)**

Funktion

Die Anweisung funktioniert genau wie die oben beschriebene ioStatus-Anweisung, liefert aber zusätzlich den Beschreibungstext und den Link (physische Adresse) für den spezifizierten Ein-/Ausgang.

Die Beschreibung ist ein freier, mit den Werkzeugen zur Konfiguration der Ein-/Ausgänge definierter Text. Das Format des physischen Links ist vom Ein-/Ausgangsgerät abhängig. Üblicherweise hat es folgende Form: 'deviceName\moduleName\ioAddress'.

Beispiel

Dieses Programm testet ein Signal und zeigt Fehlermeldungen an, wenn es nicht funktioniert.

```
if ioStatus(aiSignal, sDecription, sPath)<0  
  sOutput="Signal "+sPath+" Fehler"+"Beschreibung:"+sDecription  
endif
```

Siehe auch

num ioStatus(ai ailnputOutput)
num ioStatus(dio dilnputOutput)

3.7. TYP SIO

3.7.1. DEFINITION

Der **sio**-Typ dient zur Verbindung einer **VAL 3**-Variable mit einem seriellen Port oder einem Ethernet Socket Anschluss. Ein **sio**-Ein-/Ausgang ist gekennzeichnet durch:

- Im System definierte, dem Typ der Kommunikation eigene Parameter
- Ein Zeichenketten-Endzeichen, um die Verwendung des Typs **string** zu ermöglichen
- Eine Kommunikations-Wartefrist

Die seriellen Ein-/Ausgänge des Systems sind ständig aktiviert. Die Ethernet Socket Anschlüsse werden beim ersten Lese- oder Schreibzugriff durch ein **VAL 3**-Programm geöffnet. Die Ethernet Socket Anschlüsse werden automatisch geschlossen, wenn die **VAL 3**-Applikation geschlossen wird.

Anweisungen, in denen eine Variable des Typs **sio** verwendet wird, die nicht einem im System deklarierten Ein-/Ausgang zugewiesen wurde, bewirken einen Laufzeitfehler. Variablen des Typs **sio** werden vorprogrammiert auf einen undefinierten Link initialisiert. Der Link einer **sio**-Variablen kann von anderen **sio**-Variablen aus initialisiert werden, vom **MCP** aus oder im **VAL 3 Studio** aus der **Stäubli Robotics Suite**.

3.7.1.1. SOCKETS

Die Kommunikation über Ethernet wird durch das System über TCP/IP- oder UDP-Protokolle unterstützt. TCP/IP ermöglicht eine zuverlässige Bereitstellung eines Streams, während UDP eine unzuverlässige, aber schnellere Bereitstellung eines Streams ermöglicht.

Die Ethernet-Kommunikation verwendet **Sockets**, die an der Systemsteuerung des Roboter-Controllers deklariert werden. Sockets sind Controller-IO-Ressourcen, bei denen **sio** **VAL 3**-Variablen verbunden (**sioLink**) werden können, damit die **VAL 3**-Applikationen auf Ethernet IP kommunizieren können.

3.7.1.2. SOCKET-PARAMETER

IP-Adresse

Sie definiert eine IP-Adresse mit einer Bedeutung, die vom Socket-Typ abhängt.

TCP/ Client: Definiert die IP-Adresse des Servers, der an diese Client-Socket angeschlossen werden soll.

TCP/IP-Server: Definiert die IP-Adresse der Clients, die an diese Server-Socket angeschlossen werden sollen.

UDP

- Eine gültige Adresse: Diese UDP-Socket kann nur mit dieser IP-Adresse kommunizieren (Schreiben und Lesen).
- 0.0.0.0: Die Socket empfängt alle Meldungen von einem UDP-Client, der an diesem Port angeschlossen ist. Es wird darauf hingewiesen, dass ein Schreibvorgang an diese UDPSocket einen Laufzeitfehler 122 generiert.
- *.*.255.255: Die Socket überträgt und empfängt die Frames nur als Broadcast, das in der IP-Maske definiert ist.

Timeout

Zeitwert in Sekunden zur Verwaltung des Netzwerkzugangs (Lesen, Schreiben).

- 0 : auf den geforderten Vorgang warten.
- >0 : der Vorgang (Lesen oder Schreiben) kann bis zum Timeout in Sekunden durchgeführt werden.
- -1 : der Vorgang (Lesen oder Schreiben) wartet nicht (kein Timeout) bis zur Ausführung.

Port

32-Bit numerischer Wert zur Definition der IP-Port-Nummer.

Zeichenkettenbegrenzer

ASCII Code für das Zeichenkettenende mit **sio '='** Operatoren verwenden (im Bereich [0, 255]).

Nagle

Dieser Boole'sche Parameter optimiert die Antwortzeit der Socket-Kommunikation.

Das Ausschalten von nagle verbessert die Antwortzeit steigert aber die Netzwerkbelastrung.

3.7.1.3. TCP (TRANSMISSION CONTROL PROTOCOL)

TCP ist ein Protokoll, das das Internet Protocol (IP) ergänzt; daher wird die gesamte Folge gemeinhin mit TCP/IP bezeichnet. TCP ermöglicht eine zuverlässige, bestellte Bereitstellung eines Byte-Stream von einem Programm auf einem Computer zu einem anderen Programm auf einem anderen Computer.

Andere Applikationen, die keinen zuverlässigen Datenstream-Service erfordern, können das User Datagram Protocol (UDP) verwenden, das einen Datagram-Service bereitstellt, der der reduzierten Latenz Vorrang vor der Zuverlässigkeit gibt.

Für einen TCP/IP-Austausch muss eine Verbindung zwischen einem TCP/IP-Client und einem TCP/IP-Server hergestellt werden. Die TCP/IP-Verbindungen werden automatisch durch das System gemäß den Parametern verwaltet, die während der socket-Deklaration festgelegt wurden.

Die Socket-Verbindungen werden genau zu dem Zeitpunkt erstellt, wenn ein **VAL 3**-Programm sie benötigt (sioGet, sioSet, clearBuffer):

- Die Client-Sockets versuchen, eine Verbindung zu den deklarierten Servern herzustellen. Es wird darauf hingewiesen, dass es nur eine Instanz einer Client-Verbindung gibt. Sie wird von allen sio **VAL 3**-Variablen, die damit verbunden sind, geteilt.
- Die Server-Sockets warten auf eine Verbindung zum Client. Es wird darauf hingewiesen, dass jede mit einem Socket-Server verlinkte sio-Variable ihre eigene Verbindungsinstanz benutzt.

Die Socket-Verbindung wird automatisch geschlossen, wenn sie nicht mehr von einem **VAL 3**-Programm benutzt wird:

- Wenn die Applikation beendet wird.
- Wenn ein Fehler beim Lesen oder Schreiben festgestellt wird (mit Ausnahme eines Timeout).
- Durch die Funktion **clearBuffer**.
- Eine Server-Verbindung wird geschlossen, wenn seine eigene **VAL 3**-Variable gelöscht wird (lokale Variablen werden beim Verlassen der Routine gelöscht).
- Eine Client-Verbindung wird geschlossen, wenn keine **VAL 3**-Variable mehr darauf Bezug nimmt (eine Bibliothek entladen).

3.7.1.4. UDP (USER DIAGRAM PROTOCOL)

UDP verwendet ein einfaches Übertragungsmodell ohne implizite Handshaking-Dialoge für Zuverlässigkeit, Anforderung oder Datenintegrität. Daher liefert UDP einen unzuverlässigen Service und die Datagrams können in anderer Reihenfolge ankommen, dupliziert erscheinen oder ohne Vorankündigung verloren gehen.

UDP nimmt an, dass die Überprüfung und Behebung des Fehlers entweder nicht erforderlich ist oder in der Applikation vorgenommen wird, wodurch der Aufwand einer solchen Verarbeitung auf Netzwerk-Interface-Level vermieden wird. Zeitkritische Applikationen verwenden oft UDP, da das Ablegen von Paketen dem Warten auf verzögerte Pakete vorgezogen wird, was in einem Echtzeitsystem keine Option darstellt.

Verglichen mit TCP, sind UDP Sockets nicht als Client oder Server gekennzeichnet, unterstützen aber beide: Die gleiche Socket kann zum Lesen und zum Schreiben verwendet werden.

Wenn Fehlerbehebungsfunktionen auf Netzwerk-Interface-Level erforderlich sind, sollte vorzugsweise das bereits vorhandene, zu diesem Zweck entwickelte Transmission Control Protocol (TCP) benutzt werden.

3.7.2. OPERATOREN

Wird der Timeout beim Lesen oder Schreiben eines seriellen Ein-/Ausgangs erreicht, so wird ein Laufzeitfehler erzeugt.

| | |
|---|--|
| string <sio siOutput> = <string sText> | Schreibt nacheinander in siOutput die Unicode UTF8 -Codes von sText Zeichen, gefolgt vom Zeichenketten-Endzeichen, und übergibt sText . |
| num <sio siOutput> = <num nData item> | Schreibt in siOutput die nData item nächstliegende ganze Zahl, Modulo 256 , und über gibt den tatsächlich übermittelten Wert. |
| num <num nData> = <sio silnput> | Liest ein Byte in silnput und ordnet nData dem Byte-Wert zu. |
| string <string sText> = <sio silnput> | Liest in silnput eine Unicode UFT8 Zeichenkette und ordnet dieser sText zu. Mit dem Typ string nicht kompatible Zeichen werden außer Acht gelassen. Die Zeichenkette endet, wenn das Zeichenketten-Endzeichen gelesen wird, oder wenn sText die maximale Größe des Typs string (128 Byte) erreicht hat. Das Zeichenketten-Endzeichen wird nicht in sText übertragen. |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig. **nLen=len(sString=silnput)** muss ersetzt werden durch **sString=silnput; nLen=len(sString)**.

3.7.3. ANWEISUNGEN

void **sioLink**(**sio& siVariable**, **sio siSource**)

Funktion

Diese Anweisungen weisen **siVariable** dem Ein-/Ausgang des seriellen Systems zu, an dem **siSource** angeschlossen ist.

Siehe auch

void dioLink(dio& diVariable, dio diSource)
void aioLink(aio& aiVariable, aio aiSource)

num **clearBuffer**(**sio siVariable**)

Funktion

Diese Anweisung leert den Lesepuffer von **siVariable** und teilt die Anzahl der so gelöschten Zeichen mit

Für eine Ethernet Socket Verbindung deaktiviert (schließt) **clearBuffer** die Socket. **clearBuffer** gibt **-1** zurück, wenn Socket bereits deaktiviert worden ist.

Ein Laufzeitfehler wird generiert, wenn **siVariable** nicht an eine serielle Systemverbindung oder EthernetSocket angeschlossen ist.

num **ioStatus**(**sio silnputOutput**)

Funktion

Diese Anweisung liefert eine positive Zahl, wenn die angegebene Ein-/Ausgangsvariable funktioniert und eine negative Zahl, wenn ein Fehler vorliegt. Der zurückgegebene Wert detailliert den Status des Ein-/Ausgangs:

| | |
|----|---|
| 0 | Der Ein-/Ausgang funktioniert. |
| -1 | Der Ein-/Ausgang funktioniert nicht, weil der Link (physische Adresse) nicht definiert ist. |
| -2 | Der Ein-/Ausgang funktioniert nicht, weil der Link (physische Adresse) keinem Ein-/Ausgangssystem entspricht. |

num **ioStatus**(**sio silnputOutput**, **string& sDescription**, **string& sPhysicalPath**)

Funktion

Die Anweisung funktioniert genau wie die oben beschriebene ioStatus-Anweisung, liefert aber zusätzlich den Beschreibungstext und den Link (physische Adresse) für den angegebenen Ein-/Ausgang.

Die Beschreibung ist ein freier, mit den Werkzeugen zur Konfiguration der Ein-/Ausgänge definierter Text. Das Format des physischen Links ist vom Ein-/Ausgangsgerät abhängig. Üblicherweise hat es folgende Form:
'deviceName\moduleName\ioAddress'

Beispiel

Dieses Programm testet ein Signal und zeigt Fehlermeldungen an, wenn es nicht funktioniert.

```
if iostatus(siSignal, sDescription, sPath)<0
    sOutput="Signal "+sPath+ "Fehler"+"Beschreibung:"+sDescription
endif
```

num **sioGet**(**sio** siInput, **num&** nData[])

Funktion

Diese Anweisung liest ein einzelnes Zeichen oder ein Array von Zeichen von **siInput** und liefert die Anzahl der gelesenen Zeichen.

Der Lesevorgang wird beendet, wenn das Array **nData** voll oder der Eingangs-Lesepuffer leer ist.

- Bei einer Ethernet Socket Verbindung versucht **sioGet** zunächst, eine Verbindung zu eröffnen, wenn es noch keine eröffnete Verbindung gibt. Nach dem Erreichen der Kommunikations-Wartefrist eines Eingangs meldet **sioGet -1**.
- Wenn die Verbindung eröffnet ist, aber keine Daten im Eingangslesepuffer vorhanden sind, wartet **sioGet**, bis Daten empfangen werden, und gibt die Größe der gelesenen Daten zurück, oder wartet bis zum Ende des Timeout und gibt dann 0 zurück. Es wartet weiter, wenn die Socket-Timeout-Parameter auf 0 gesetzt sind. Wenn ein Fehler erfasst oder eine Verbindung geschlossen wird, wird die Funktion sofort beendet und gibt -1 zurück.

Wenn der Socket-Timeout-Parameter -1 ist, erhält die Funktion den aktuellen Lesepuffer und gibt die Anzahl der gelesenen Zeichen zurück (0 wenn der Lesepuffer leer ist) oder -1 bei Lesefehler.

- Der Laufzeitfehler 123 wird generiert, wenn **siInput** nicht mit einem seriellen Systemanschluss oder EthernetSocket verbunden ist.
- Die Laufzeitfehler 20 oder 21 treten auf, wenn **nData** nicht mit dem richtigen Index benutzt wird.

num **sioSet**(**sio** siOutput, **num&** nData[],**nNbElements**)

Funktion

Diese Anweisung schreibt ein oder mehrere Zeichen in **siOutput** und gibt die Anzahl der geschriebenen Zeichen zurück.

Der optionale Parameter **nNbElements** legt die Anzahl der zu schreibenden Zeichen fest.

Die digitalen Werte werden vor der Übertragung in ganze Zahlen zwischen **0** und **255** umgewandelt, dabei wird jeweils die nächstliegende ganze Zahl Modulo **256** verwendet.

Bei einer Ethernet Socket Verbindung versucht **sioSet** zunächst, eine Verbindung zu eröffnen, wenn es noch keine eröffnete Verbindung gibt. Nach dem Erreichen der Kommunikations-Wartefrist eines Ausgangs meldet **sioSet -1**. Die Anzahl der geschriebenen Zeichen kann unter der Größe von **nData** liegen, wenn ein Kommunikationsfehler entdeckt wurde.

Der Laufzeitfehler 123 wird generiert, wenn **siOutput** nicht mit einem seriellen Systemanschluss oder EthernetSocket verbunden ist.

- Der Laufzeitfehler 20 wird generiert, wenn **nNbElements** größer als die Array-Größe **nData** ist.
- Die Laufzeitfehler 20 oder 21 treten auf, wenn **nData** nicht mit dem richtigen Index benutzt wird.
- Bei der UDP Socket wird der Laufzeitfehler 122 bei einem Schreibversuch auf IP 0.0.0.0 generiert.

num **sioCtrl**(**sio** siChannel, **string** nParameter, ***value**)

Funktion

Diese Anweisung ändert einen Kommunikationsparameter des angegebenen Sio Ein-Ausgangs siChannel.

Bei seriellen Leitungen kann es vorkommen, dass einige Parameter oder Parameterwerte nicht von der Hardware unterstützt werden: Siehe hierzu das Controller-Handbuch.

Die Anweisung gibt Folgendes zurück:

| | |
|----|--|
| 0 | Der Parameter wurde erfolgreich geändert. |
| -1 | Der Parameter ist nicht definiert. |
| -2 | Der Parameterwert hat nicht den erwarteten Typ. |
| -3 | Der Parameterwert wird nicht unterstützt. |
| -4 | Der serielle Kanal ist nicht dazu bereit die Parameteränderung anzuwenden (muss zuerst gestoppt werden). |
| -5 | Der Parameter ist für diesen Kanaltyp nicht definiert. |

Die unterstützten Parameter werden in der unten stehenden Tabelle angegeben:

| Parameter-name | Parametertyp | Beschreibung |
|----------------|--------------|--|
| "port" | num | (Für TCP Client oder Server) TCP Port |
| "target" | string | (Für TCP Client) IP-Adresse des zu erreichenden TCP Servers, z.B. "192.168.0.254" |
| "clients" | num | (Für TCP Server) Maximale Anzahl gleichzeitiger Clients auf dem Server |
| "endOfString" | num | (Für serielle Leitung, TCP Client und Server) ASCII Code für das Zeichenkettenende mit sio '=' Operatoren verwenden (im Bereich [0, 255]) |
| "timeout" | num | (Für serielle Leitung, TCP Client und Server) Maximaler Timeout für den Kommunikationskanal. 0 bedeutet kein Timeout. |
| "baudRate" | num | (Für serielle Leitung) Kommunikationsgeschwindigkeit |
| "parity" | string | (Für serielle Leitung) Paritätskontrolle: "none", "even" oder "odd" |
| "bits" | num | (Für serielle Leitung) Anzahl Bits pro Byte (5, 6, 7 oder 8) |
| "stopBits" | num | (Für serielle Leitung) Anzahl Stop-Bits pro Byte (1 oder 2) |
| "mode" | string | (Für serielle Leitung) Kommunikationsmodus: "rs232" |
| "flowControl" | string | (Für serielle Leitung) Datenflusskontrolle: "none" oder "hardware" |
| "nagle" | bool | (Für TCP Client oder Server) die nagle Optimierung aktivieren (standardmäßig) oder deaktivieren. Das Ausschalten von nagle verbessert die Antwortzeit steigert aber die Netzwerkbelastung. |

Beispiel

Dieses Programm konfiguriert die wichtigsten Parameter für eine serielle Schnittstelle.

```
sioCtrl(siPortSerial1, "baudRate", 115200)
sioCtrl(siPortSerial1, "bits", 8)
sioCtrl(siPortSerial1, "parity", "none")
sioCtrl(siPortSerial1, "stopBits", 1)
sioCtrl(siPortSerial1, "timeout", 0)
sioCtrl(siPortSerial1, "endOfString", 13)
```


KAPITEL 4

BENUTZERSCHNITTSTELLE

4.1. ANWEISUNGEN

num **userPage(string sPageName)**

Funktion

Diese Anweisung zeigt auf dem **MCP**-Bildschirm die spezifizierte Benutzeroberseite an.

Der **sPageName**-Parameter ist der Name der Seite. Dieser Name entspricht eigentlich einem Dateinamen ohne Erweiterung **html**, der die Seite beschreibt.

Standardgemäß sucht die **userPage**-Funktion nach der Datei im Verzeichnis mit dem Namen "interface" des aktuellen Applikationsverzeichnisses. Ein Pfad zu der Datei kann auch anhand der **VAL 3**-Dateipfadkonventionen festgelegt werden.

Die **userPage**-Funktion gibt 0 zurück, wenn die der Seite entsprechende Datei gefunden und fehlerfrei geladen wird, andernfalls wird -1 zurückgegeben.

Beispiel

Dieses Programm zeigt die Benutzeroberseite mit dem Namen **myUserPage** an, die der Datei **myUserPage.html** in der Verzeichnisschnittstelle der Applikation entspricht.

```
nRet=userPage( "myUserPage" )
```

```
void userPageBind (string sPage, string sGraphicalObjectId,
    string sGraphicalObjectAttrib, num& nVar, num nSize,
    string sDirection, num nRefresh)
```

```
void userPageBind (string sPage, string sGraphicalObjectId,
    string sGraphicalObjectAttrib, string& nVar, num nSize,
    string sDirection, num nRefresh)
```

```
void userPageBind (string sPage, string sGraphicalObjectId,
    string sGraphicalObjectAttrib, bool& nVar, num nSize,
    string sDirection, num nRefresh)
```

```
void userPageBind (string sPage, string sGraphicalObjectId,
string sGraphicalObjectAttrib, dio& dDio, num nSize, string sDirection,
    num nRefresh)
```

```
void userPageBind (string sPage, string sGraphicalObjectId,
string sGraphicalObjectAttrib, aio& aAio, num nSize, string sDirection,
    num nRefresh)
```

Funktion

Diese Funktion bindet das Attribut **sGraphicalObjectAttrib** des grafischen Objekts **sGraphicalObjectId** der Seite **sPage** an eine **VAL 3**-Variable.

| | |
|-------------------------------|---|
| sPage | Die Seite enthält das zu bindende grafische Element (Gleiche Zeichenkette wie die in der userPage() -Funktion benutzte). |
| sGraphicalObjectId | Die einmalige ID des zu bindenden grafischen Elements. |
| sGraphicalObjectAttrib | Das Attribut des zu bindenden grafischen Elements. |
| nVar | Ein Element einer Variable. |
| nSize | Anzahl der zu bindenden Elemente. |

| sDirection | Definiert die Richtung des Bindens von der grafischen Seite aus gesehen. | |
|-------------------|---|--|
| | Werte | Beschreibung |
| | r | Lesen. Das grafische Element wird mit dem Wert der VAL 3 -Variable aktualisiert. |
| | w | Schreiben. Das grafische Element aktualisiert den Wert der VAL 3 -Variable. |
| | rw | Lesen/Schreiben. Das grafische Element wird mit dem Wert der VAL 3 -Variable aktualisiert und aktualisiert den Wert der VAL 3 -Variable. |
| | r1 | Einmal Lesen. Das grafische Element wird mit dem Wert der VAL 3 -Variable nur einmal aktualisiert, wenn die Seite geladen wird. |
| nRefresh | Einmal Lesen/Schreiben. | |
| | Optionaler Parameter, der die Aktualisierungszykluszeit festlegt. 1 für jeden Zyklus, 2 für einen Zyklus auf 2, Dieser Parameter ist nur nützlich, wenn die Richtung entweder "r" oder "rw" ist. | |

Anmerkungen zum Umfang von beschränkten Variablen

Es gibt einige Einschränkungen für beschränkte Variablen. Diese Variable kann Folgendes sein:

- Eine globale Variable,
- Ein Teil einer strukturierten Variable wie `joint` oder `point`,
- Ein Teil einer benutzerdefinierten Struktur,
- Eine Variable aus einer anderen Bibliothek,
- Ein Funktionsparameter, wenn dieser Parameter ein Bezugsparameter ist,
- Die Bindegöße kann nur > 1 sein, wenn der Container-Typ ein Array ist.

Beispiel

```
// Fill select options with firsts elements of val3_array
userPageBind("pagel","select_id1","options",val3_array[0],5,"r1")

// get the selected value in the variable val_string
userPageBind("pagel","select_id1","value",val_string,1,"w")

// Force the selection to the option "hello"
val_string = "hello"
userPageBind("pagel","select_id1","value",val_string,1,"r")
```

**bool userPageUnbind(string sPage, string sGraphicalObjectId,
 string sGraphicalObjectAttrib)**

Funktion

Diese Funktion entfernt die Bindung des Attributs **sGraphicalObjectAttrib** des grafischen Objekts **sGraphicalObjectId** der Seite **sPage**.

Die Funktion gibt **false** zurück, wenn die Bindung nicht existiert. Dies ist nicht verbindlich zum Prüfen des Rückgabewerts.

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, bool& bVar, bool& bValue)**

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, dio& bVar, bool& bValue)**

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, num& nVar, num& nValue)**

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, aio& aVar, num& bValue)**

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, string& sVar, string& sValue)**

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, bool& bVar)**

**void userPageBindXEVENTX(string sPage,
string sGraphicalObjectId, dio& dVar)**

Funktion

Diese Funktion bindet das Ereignis **XEVENTX** des grafischen Objekts sGraphicalObjectId der Seite sPage an eine **VAL 3**-Variable.

Die **VAL 3**-Variable wird jedes Mal auf **sValue** gesetzt, wenn das Ereignis **XEVENTX** ausgelöst wird.

Für **bool** und **dio** gibt es einen anderen Prototypen ohne den Wertparameter. In diesem Fall wird die Variable jedes Mal umgeschaltet, wenn das Ereignis ausgelöst wird.

| XEVENTX | Beschreibung |
|-----------|---|
| Mouseup | Gesendet, wenn der Benutzer ein vorher gedrücktes grafisches Element frei gibt. |
| Mousedown | Gesendet, wenn der Benutzer ein grafisches Element drückt. |
| Click | Gesendet, wenn ein grafisches Element angeklickt wird. |

**bool userPageUnbindXEVENTX(string sPage,
string sGraphicalObjectId, bool& bVar)**

**bool userPageUnbindXEVENTX(string sPage,
string sGraphicalObjectId, dio& bVar)**

**bool userPageUnbindXEVENTX(string sPage,
string sGraphicalObjectId, num& nVar)**

**bool userPageUnbindXEVENTX(string sPage,
string sGraphicalObjectId, aio& aVar)**

**bool userPageUnbindXEVENTX(string sPage,
string sGraphicalObjectId, string& sVar)**

Funktion

Diese Funktion löscht die Verbindung des Ereignisses **XEVENTX** des grafischen Objekts **sGraphicalObjectId** der Seite **sPage** zur **VAL 3**-Variable.

Ein Ereignis kann mehreren Verbindungen zugeordnet werden.

Diese Funktion kehrt zu wahr zurück, wenn eine zugeordnete Verbindung gefunden und gelöscht worden ist.

| XEVENTX | Beschreibung |
|-----------|---|
| Mouseup | Gesendet, wenn der Benutzer ein vorher gedrücktes grafisches Element frei gibt. |
| Mousedown | Gesendet, wenn der Benutzer ein grafisches Element drückt. |
| Click | Gesendet, wenn ein grafisches Element angeklickt wird. |

Beispiel

Verwendung einer Boole'schen Variable zur Erstellung einer Callback-Funktion.

clickCallback.pgx

```

begin
  x_nCounter = 0
  while true
    // setMutex() wait for x_bClicked to be false, then set it to true and exit
    setMutex(x_bClicked)
    x_nCounter = x_nCounter + 1
  endWhile
end

```

start.pgx

```
...
// Init
bButtonClicked = true
nButtonClickCounter = 0

// Create callback function
taskCreate "tClickCb", 10, clickCallback(bButtonClicked, nButtonClickCounter)

// Force bButtonClicked to false each time some_button is clicked
userPageBindClick("index", "some_button", bButtonClicked, false)

...
...
```

Ändern des Textes einer Schaltfläche je nach Status

```
...
userPageBindMousedown("some_page", "some_button", sButtonState, "PUSHED")
userPageBindMouseup("some_page", "some_button", sButtonState, "RELEASED")
userPageBind("some_page", "some_button", "innerHTML", sButtonState, 1, "r")

...
```

```
void userPageSet(string sPage, string sGraphicalObjectId,
                 string sGraphicalObjectAttrib, bool nVar)
void userPageSet(string sPage, string sGraphicalObjectId,
                 string sGraphicalObjectAttrib, num nVar)
void userPageSet(string sPage, string sGraphicalObjectId,
                 string sGraphicalObjectAttrib, string nVar)
```

Funktion

Diese Funktion bindet das Attribut **sGraphicalObjectAttrib** des grafischen Objekts **sGraphicalObjectId** der Seite **sPage** statisch an eine **VAL 3**-Variable oder einen konstanten Wert.

| | |
|-------------------------------|---|
| sPage | Die Seite enthält das zu bindende grafische Element (Gleiche Zeichenkette wie die in der userPage() -Funktion benutzte). |
| sGraphicalObjectId | Die einmalige ID des zu bindenden grafischen Elements. |
| sGraphicalObjectAttrib | Das Attribut des zu bindenden grafischen Elements. |
| nVar | Eine VAL 3 -Variable oder ein konstanter Wert. |

Anmerkungen zum Umfang von beschränkten Variablen

Es gibt einige Einschränkungen für beschränkte Variablen. Diese Variable kann Folgendes sein:

- Eine globale Variable,
- Ein Teil einer strukturierten Variable wie **joint** oder **point**,
- Ein Teil einer benutzerdefinierten Struktur,
- Eine Variable aus einer anderen Bibliothek,
- Ein Funktionsparameter, wenn dieser Parameter ein Bezugsparameter ist.

Beispiel

```
// Set label text directly
userPageSet("index", "myLabel", "innerTHML", "Hello, world!")

// Hide a button (you can use local variables here)
l_sVisibility = "hidden"
userPageSet("index", "myButton", "style.visibility", l_sVisibility)

// The following line won't make the button visible because l_sVisibility is not bound
l_sVisibility = "visible"

// Disable a button with a boolean value
userPageSet("index", "myButton", "disabled", true)
```

void userPageAlert(string sMessage)

Funktion

Anzeige von **sMessage** in einem Feld und warten, bis die Schaltfläche OK gedrückt wird.

bool userPageConfirm(string sMessage)

Funktion

Anzeige von **sMessage** in einem Feld und Rückgabe von True, wenn die Schaltfläche OK gedrückt wird, und False, wenn die Schaltfläche CANCEL gedrückt wird.

bool userPagePrompt(string sMessage, string& sEntry)

Funktion

Den Benutzer zu einer Eingabe mit **sMessage** auffordern. Die Eingabe wird an **sEntry** zurückgegeben.

Rückgabe von True, wenn die Schaltfläche OK gedrückt wird, und False, wenn die Schaltfläche CANCEL gedrückt wird.

bool userPagePrompt(string sMessage, num& nNum)

Funktion

Den Benutzer zu einer Eingabe mit **sMessage** auffordern. Die Eingabe wird an **sEntry** zurückgegeben.

Gibt True zurück, wenn die Schaltfläche OK gedrückt wird, und False, wenn die Schaltfläche CANCEL gedrückt wird.

**void popUpMsg(string sText),
void popUpMsg(string sText, num nSeverity)**

Funktion

Diese Anweisung zeigt **sText** in einem "popup"-Fenster über dem aktuellen **MCP**-Fenster an. Dieses Fenster bleibt so lange angezeigt, bis es im Menü mit **Ok** bestätigt oder die **Esc**-Taste gedrückt wird.

Der optionale Parameter **nSeverity** ermöglicht die Festlegung einer Prioritätsstufe für die Meldung in der **VAL 3**-Applikation. Folgende Werte werden unterstützt:

| sPage | Bedeutung | Beschreibung |
|--------------|-------------------|--|
| 1 | Information | Meldungen zum Normalbetrieb, die keine Aktion erfordern. |
| 2 | Achtung | Kann darauf hinweisen, dass ein Fehler auftreten wird, wenn nichts unternommen wird. |
| 3 | Fehler | Fehlerbedingungen. |
| 4 | Kritischer Fehler | Kritische Bedingungen. |

Siehe auch

num userPage(string sPageName)

**bool logMsg(string sText),
bool logMsg(string sText, num nSeverity)**

Funktion

Diese Anweisung schreibt **sText** in das Logfile des Systems (Fehlerprotokoll). Die Meldung wird mit Datum und Uhrzeit registriert. "USR" wird dem Anfang der Zeichenkette hinzugefügt, um sie als Benutzernachricht zu kennzeichnen. Wenn viele Log-Meldungen gleichzeitig protokolliert werden, können einige verloren gehen. Die Anweisung liefert dann 'false' zurück, damit die Meldungen dann später geschrieben werden können, falls erforderlich.

Der optionale Parameter **nSeverity** ermöglicht die Festlegung einer Prioritätsstufe für die Meldung in der **VAL 3**-Applikation. Folgende Werte werden unterstützt:

| nSeverity Wert | Bedeutung | Beschreibung |
|-----------------------|-------------------|--|
| 1 | Information | Meldungen zum Normalbetrieb, die keine Aktion erfordern. |
| 2 | Achtung | Kann darauf hinweisen, dass ein Fehler auftreten wird, wenn nichts unternommen wird. |
| 3 | Fehler | Fehlerbedingungen. |
| 4 | Kritischer Fehler | Kritische Bedingungen. |

Beispiel

Dieses Programm stellt sicher, dass die Meldung protokolliert wird:

```
while logMsg(sMessage)==false
delay(0)
endWhile
```

Siehe auch

void popUpMsg(string sText), void popUpMsg(string sText, num nSeverity)

string getProfile()

Funktion

Diese Anweisung gibt den Namen des aktuellen Benutzerprofils zurück.

Siehe auch

num setProfile(string sUserLogin, string sUserPassword)

num setProfile(string sUserLogin, string sUserPassword)

Funktion

Diese Anweisung ändert das aktuelle Benutzerprofil (sofortige Wirkung).

Die Funktion liefert folgende Antwort:

| | |
|----|---|
| 0 | Das angegebene Benutzerprofil ist jetzt wirksam. |
| -1 | Das angegebene Benutzerprofil ist nicht definiert. |
| -2 | Das angegebene Benutzerpasswort ist nicht richtig. |
| -3 | Das Benutzerprofil ' staubli ' ist mit dieser Anweisung nicht zulässig. |
| -4 | Das aktuelle Benutzerprofil ist ' staubli ' und kann mit dieser Anweisung nicht geändert werden. |

Siehe auch

string getProfile()

string getLanguage()

Funktion

Diese Anweisung gibt die aktuelle Spracheinstellung des Controllers zurück.

Beispiel

```
switch(getLanguage())
  case "francais"
    sMessage="Attention!"
  break
  case "english"
    sMessage="Warning!"
  break
  case "deutsch"
    sMessage="Achtung!"
  break
  case "italiano"
    sMessage="Avviso!"
  break
  case "espanol"
    sMessage="¡Advertencia!"
  break
default
  sMessage="Warning!"
break
endSwitch
```

Siehe auch

[bool setLanguage\(string sLanguage\)](#)

bool setLanguage(string sLanguage)

Funktion

Diese Anweisung ändert die aktuelle Sprache des Controllers: der Name der angegebenen Sprache sLanguage muss dem Namen einer Übersetzungsdatei im Controller entsprechen. Schauen Sie im Controller-Handbuch nach, um zusätzliche Sprachen auf dem Controller zu installieren oder von dort zu entfernen.

Beispiel

Dieses Programm schaltet die Controllersprache auf Chinesisch um:

```
if(setLanguage("chinese") == false)
  sOutput="The Chinese language is not available on the robot controller"
endif
```

Siehe auch

[string getLanguage\(\)](#)

string getDate(string sFormat)

Funktion

Diese Anweisung liefert das aktuelle Datum und/oder Uhrzeit des Controllers. Im Parameter sFormat wird das Format für die übertragenen Daten angegeben. In diesem String wird jedes Vorkommen von Schlüsselworten durch den entsprechenden Datums- oder Zeitwert ersetzt. Die unterstützten Format-Schlüsselwörter werden in der unten stehenden Tabelle aufgelistet:

| Schlüsselwort | Beschreibung |
|---------------|---|
| %y | 2-stellige Jahresangabe (00-99), ohne Jahrhundert |
| %Y | 4-stellige Jahresangabe, wie z.B. 2007 |
| %m | Monat (00-12) |
| %d | Tag (00-31) |
| %H | Stunden im 24-Stunden-Format (00-23) |
| %I | Stunden im 12-Stunden-Format (01-12) |
| %p | A.M./P.M. Angabe für 12-Stunden-Uhr |
| %M | Minuten (00-59) |
| %S | Sekunden (00-59) |

Beispiel

Dieses Programm zeigt Datum und Uhrzeit im Format "January 01, 2007 13:45:23" an

```
switch (getDate( "%m" ))
    case "01"
        sMonth="January"
    break
    case "02"
        sMonth="February"
    break
    case "03"
        sMonth="March"
    break
    case "04"
        sMonth="April"
    break
    case "05"
        sMonth="May"
    break
    case "06"
        sMonth="June"
    break
    case "07"
        sMonth="July"
    break
    case "08"
        sMonth="August"
    break
    case "09"
        sMonth="September"
    break
    case "10"
        sMonth="October"
    break
    case "11"
        sMonth="November"
    break
    case "12"
        sMonth="December"
```

```
break
default
sMonth="???"
break
endSwitch
// Display date and date in the form: "January 01, 2007 13:45:23"
sOutput=getDate(sMonth+" %d, %Y %H:%M:%S")
```

KAPITEL 5

TASKS

5.1. DEFINITION

Eine Task ist ein Programm, das zu einem gegebenen Zeitpunkt ausgeführt wird. Eine Applikation kann und wird im Allgemeinen mehrere Tasks in Ausführung haben.

In einer Applikation befinden sich typischerweise Tasks für die Armbewegungen, eine Task Automat, eine Task für die Benutzerschnittstelle, eine Task für die Überwachung der Sicherheitssignale, Kommunikationstasks usw.

Eine Task ist durch folgende Elemente gekennzeichnet:

- Namen: eine eindeutige Kennung innerhalb der Bibliothek oder Applikation
- eine Priorität oder Periode: Parameter für Ablaufsteuerung der Tasks
- Programm: Einsprungstelle (und Aussprungstelle) der Task
- Status: aktiv oder beendet
- die nächste auszuführende Anweisung (mit Kontext)

5.2. FORTSETZEN NACH EINEM LAUFZEITFEHLER

Wenn eine Anweisung zu einem Laufzeitfehler geführt hat, wird die Task unterbrochen. Mit der Anweisung `taskStatus()` kann die Fehlerursache ermittelt werden. Anschließend wird die Task mit der Anweisung `taskResume()` fortgesetzt. Wenn die Störungsursache behoben werden konnte, macht das Programm in der Zeile weiter, in der die Task unterbrochen wurde. Andernfalls muss vor oder nach dieser Zeile begonnen werden.

Starten und Beenden der Applikation

Zum Starten einer Applikation wird ihr Programm `start()` in einer Task mit dem Namen der Applikation, dem nachgestellten Zeichen '~' und der Priorität **10** ausgeführt.

Zum Beenden einer Applikation wird ihr Programm `stop()` in einer Task mit dem Namen der Applikation, dem ein '~' vorangestellt ist, und der Priorität **10** ausgeführt.

Wird eine **VAL 3**-Applikation von der Benutzerschnittstelle der **MCP** aus unterbrochen, so wird die Start-Task, falls sie noch vorhanden ist, augenblicklich gelöscht. Dann wird das Programm `stop()` ausgeführt, danach alle noch nicht ausgeführten Tasks der Applikation in der umgekehrten Reihenfolge ihrer Erstellung gelöscht und schließlich werden Bibliotheken aus dem Speicher entfernt.

5.3. SICHTBARKEIT

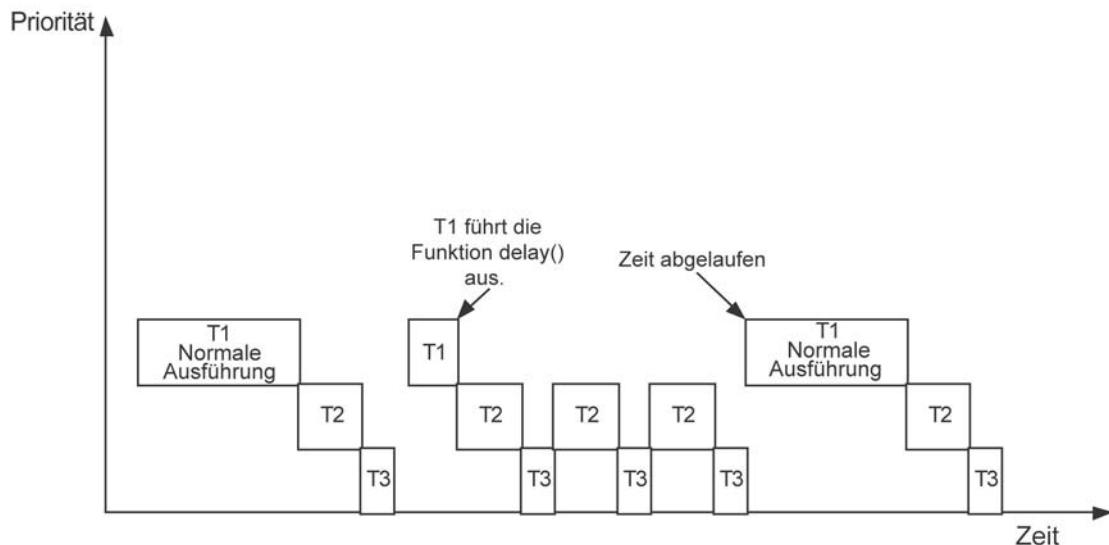
Die Tasks sind nur innerhalb des Programms oder der Bibliothek sichtbar, von der sie erstellt wurden. Die Anweisungen `taskSuspend()`, `taskResume()`, `taskKill()` und `taskStatus()` wirken auf eine von einer anderen Bibliothek erstellte Task so, als wäre die Task nicht erstellt worden. Zwei verschiedene Bibliotheken können also Tasks mit identischen Namen erstellen.

5.4. SEQUENTIELLES ORDNEN

Wenn in einer Applikation mehrere Tasks ausgeführt werden, hat es den Anschein, als ob dies gleichzeitig und unabhängig voneinander erfolgt. Das ist jedoch nur richtig, wenn man die Applikation global, d.h. über genügend lange Zeitintervalle von z.B. einer Sekunde betrachtet. Bei kurzen Zeitintervallen ist diese Aussage nicht mehr gültig.

Da das System nur einen Prozessor besitzt, kann es auch nur jeweils eine Task ausführen. Die Gleichzeitigkeit der Ausführung wird durch ein rasches sequentielles Ordnen der einzelnen Tasks erreicht, die reihum einige Anweisungen abarbeiten, bevor das System auf die nächste Task weitergeht.

Sequentielles Ordnen



Unter **VAL 3** sind die Sequenzen der Tasks nach folgenden Regeln geordnet:

1. Die Tasks werden in der Reihenfolge ihrer Erstellung sequentiell geordnet.
2. In jeder Sequenz versucht das System, die Anzahl von **VAL 3**-Programmzeilen auszuführen, die der Priorität der Task entspricht.
3. Kann eine Programmzeile nicht beendet werden (Laufzeitfehler, Warten auf Signal, Task unterbrochen usw.), geht das System zur nächsten **VAL 3**-Task über.
4. Nach Beendigung aller **VAL 3**-Tasks bleibt dem System Zeit zur Ausführung von System-Tasks niedrigerer Priorität (z.B. Netzwerk-Kommunikation, Aktualisierung des Benutzeroberflächenbildschirms, Dateizugriff) bevor ein neuer Zyklus gestartet wird.
Die maximale Wartezeit zwischen zwei Sequenzyklen entspricht der Dauer des letzten Sequenzierungszyklus, oft jedoch ist diese Zeit gleich null, da das System sie nicht benötigt.

Folgende Anweisungen in **VAL 3** bewirken ein sofortiges Weitergehen zur Sequenz der nächsten Task:

- `watch()` (zeitlich begrenztes Warten auf eine Bedingung)
- `delay()` (Wartezeit)
- `wait()` (Warten auf eine Bedingung)
- `waitEndMove()` (Warten auf Stillstand des Roboters)
- `open()` und `close()` (Warten auf Stillstand des Roboters mit anschließender Wartezeit)
- `taskResume()` (wartet, bis die Task zum Neustart bereit ist)
- `taskKill()` (wartet, bis die Task tatsächlich gekillt ist)
- `disablePower()` (wartet, bis die Leistung tatsächlich abgeschaltet wurde)
- Die Anweisungen zum Zugriff auf den Inhalt der Festplatte (`libLoad`, `libSave`, `libDelete`, `libList`, `setProfile`)
- Die Anweisungen zum Lesen / Schreiben von sio (Bediener =, `sioGet()`, `sioSet()`)
- `setMutex()` (wartet bis boolescher Mutex falsch ist)

5.5. SYNCHRONE TASKS

Die oben beschriebene Sequenz entspricht der Sequenz der normalen, asynchronen Tasks, die das System so verwaltet, dass sie so schnell wie möglich ausgeführt werden. Manchmal muss die Abarbeitung der Tasks in einem festen Intervall ausgeführt werden, sowohl zur Datenerfassung als auch zur Steuerung von Peripheriegeräten: man spricht dann von synchronen Tasks.

Sie werden im Zyklus durch Unterbrechung der aktuellen asynchronen Task zwischen zwei **VAL 3**-Zeilen ausgeführt. Nach Beendigung der synchronen Tasks wird die asynchrone Task wieder aufgenommen.

Die Ausführung der synchronen **VAL 3**-Tasks unterliegt folgenden Regeln:

1. Jede Task wird genau einmal für jede bei der Erstellung der Task definierte Periode ausgeführt (z.B. alle 4 ms).
2. Bei jedem Zyklus führt das System bis zu 3000 **VAL 3**-Programmzeilen aus. Wenn eine Programmzeile nicht sofort beendet werden kann (Ausführungsfehler (runtime), Warten auf Signal, Task unterbrochen usw.), geht es zur nächsten Task über.
In der Praxis wird eine synchrone Task oft ausdrücklich durch die Anweisung `delay(0)` beendet, die das System zur Ausführung der nächsten Task zwingt.
3. Sychrone Tasks gleicher Dauer werden in ihrer Erstellungsreihenfolge ausgeführt.

5.6. ZEITÜBERLAUF

Übersteigt die Ausführungszeit einer synchronen **VAL 3**-Task den definierten Zeitraum, so wird der aktuelle Zyklus normal beendet, der folgende aber gelöscht. Dieser Zeitüberlauf wird der **VAL 3**-Applikation signalisiert, indem die im `taskCreate`-Aufruf angegebene boolesche Variable auf "**true**" gesetzt wird. Damit zeigt diese boolesche Variable zu Beginn jedes Zyklus an, ob die vorherige Ausführung vollständig ausgeführt wurde oder nicht.

5.7. AKTUALISIERUNG DER EIN-/AUSGÄNGE

Bezüglich der Aktualisierung der Ein-/Ausgänge gibt es 2 Gruppen von Karten:

- **Systemkartengruppe:**

CpIO, DsIO, FastIO, PowerSupplyIO, Rsi9IO, StarcIO.

- **Benutzerkartengruppe:**

J206 (EtherCAT master), J207/J208 (Real Time Ethernet), Fieldbus (Hilscher CIFX50E-xx PCIe boards).

Die Systemkartengruppe Zykluszeit aktualisieren basiert auf der Systemzykluszeit. Die Benutzerkartengruppe Zykluszeit aktualisieren kann in verschiedenen Werten der Systemzykluszeit angepasst werden.

Die Benutzergruppenkarte Aktualisieren kann in 2 verschiedenen Modi konfiguriert werden:

- Fehlermodus.
- No Jitter Modus.

Beschreibung der folgenden Zeitbilder:

| Task-Name | Beschreibung |
|-----------------|--|
| Synchro task | Diese Task verwaltet die Roboterbahn und die Aktualisierung der IO-Karten des Systems. |
| IO refresh task | Diese Task verwaltet die Aktualisierung der IO-Karten des Benutzers. |
| VAL3 synchro | Diese Task verwaltet VAL 3 synchrone Tasks (Siehe Kapitel 5.5). |
| VAL3 asynchro | Diese Task verwaltet VAL 3 Tasks (Siehe Kapitel 5.1). |

S: Steht für **System**-IO-Kartengruppe.

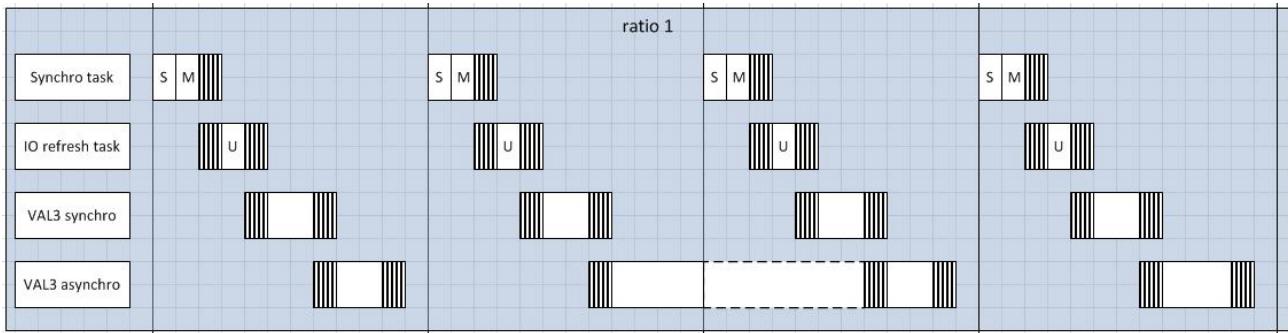
U: Steht für **User**-IO-Kartengruppe (Benutzerkartengruppe).

M: Steht für **Motion process** (Bewegungsprozess), Bahngenerator.

Zeigt den Execution-Jitter der Task an.

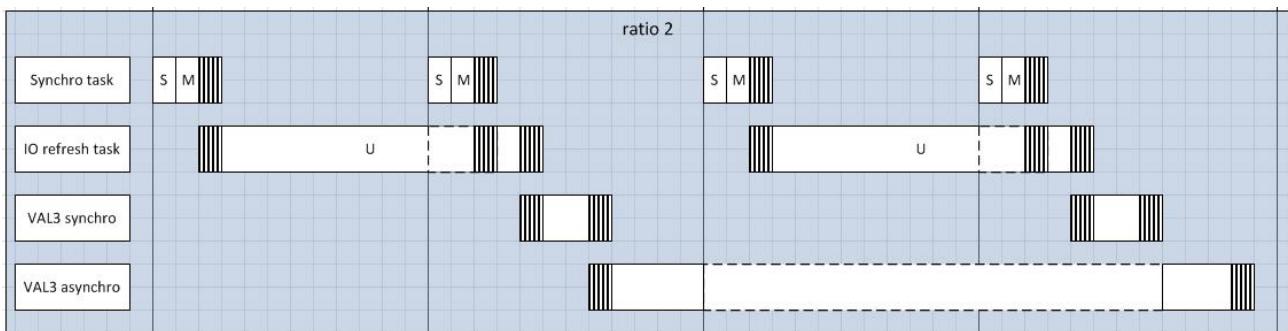
5.7.1. BENUTZERKARTEN AKTUALISIERUNG STANDARDMODUS

Die folgende Abbildung zeigt die Ausführung der verschiedenen Tasks, wenn das Aktualisierungszyklusverhältnis der **User-Board-Group auf 1** gesetzt ist.



Wenn die Aktualisierungszeit der Benutzer-IO-Karte zu lang ist (je nach Anzahl der IO), wird ein Overrun für die IO-Aktualisierungstask ausgeführt. In einer solchen Situation muss das Aktualisierungsverhältnis der Benutzer-IO/Karte erhöht werden.

Die folgende Abbildung zeigt die Ausführung der verschiedenen Tasks, wenn das Aktualisierungszyklusverhältnis der **Benutzerkarten auf 2** gesetzt ist.



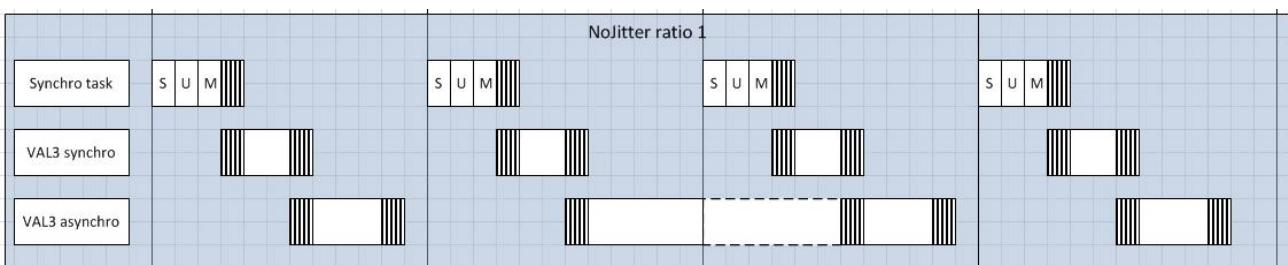
Es wird darauf hingewiesen, dass:

- sich Benutzer-IO-Werte während eines "VAL3 asynchro"-Zyklus ändern können.
- ein Overrun für eine "IO refresh task" den Roboter nicht stoppt, dafür werden aber alle Benutzer-IO auf ungültig gesetzt. Das Aktualisierungsverhältnis der Benutzerkarte kann am MCP geändert werden und erfordert keinen Neustart.

5.7.2. AKTUALISIERUNG DER BENUTZERKARTE NO JITTER MODUS

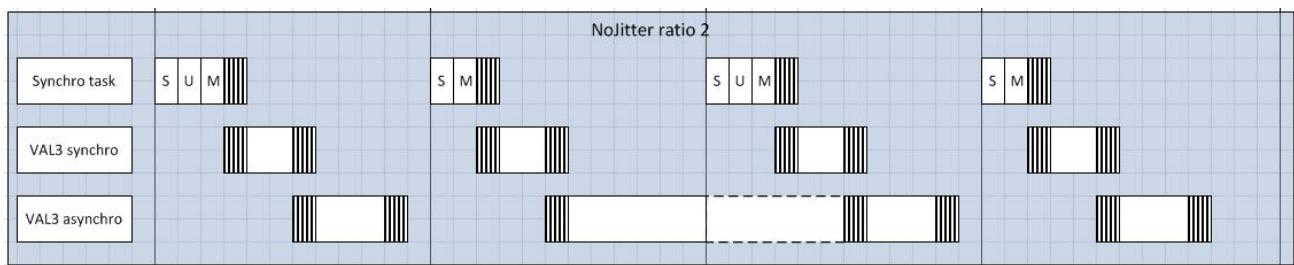
In diesem Modus gibt es keine IO-Aktualisierungstask und die Benutzerkarten werden im Rahmen der Synchro-Task aktualisiert.

Die folgende Abbildung zeigt die Ausführung der verschiedenen Tasks, wenn das Aktualisierungszyklusverhältnis der **User-Board-Group auf 1** gesetzt ist.



Wenn die Aktualisierungszeit der Benutzerkarte (U) zu lang ist (je nach Anzahl der IO), wird ein Overrun für "synchro task" durchgeführt. Dieser Overrun gilt als Fehler, **der Roboter wird gestoppt und muss neu gestartet werden.**

Die folgende Abbildung zeigt die Ausführung der verschiedenen Tasks, wenn das Aktualisierungszyklusverhältnis der **User-Board-Group** auf 2 gesetzt ist.



Es wird darauf hingewiesen, dass sich die Benutzer-IO-Werte während eines "VAL3 asynchro"-Zyklus ändern können.

5.7.3. AUSWAHL DES AKTUALISIERUNGSMODUS DER BENUTZERKARTEN UND FESTLEGUNG DES ZYKLUSVERHÄLTNISSES

Der Aktualisierungsmodus der Benutzerkarten und das Zyklusverhältnis können in der Datei usr/configs/cell.cfx geändert werden.

```
<UserIOResfresh>
  <string name = "mode" value = "MODE" />
  <Float name = "cyclicRatio" value = "RATIO" />
</UserIOResfresh>
```

| Modus | Verhältnis |
|----------|--|
| default | -1: Standardwert verwenden (auf 1 gesetzt). |
| noJitter | 1 bis N: Verhältnis in Bezug auf die Systemzykluszeit (Standardwert 4 ms). |

Beispiel für eine Standardsystemzykluszeit (4ms)

| Verhältnis | Verhältnis |
|------------|------------|
| 1 | 4 ms |
| 2 | 8 ms |
| 3 | 12 ms |
| ... | ... |

5.8. SYNCHRONISIERUNG

Manchmal müssen mehrere Tasks miteinander synchronisiert werden, bevor ihre Ausführung fortgesetzt werden kann.

Wenn die Ausführungszeiten der einzelnen Tasks bekannt sind, kann die Synchronisierung einfach durch Warten auf ein Signal des langsamsten Jobs erfolgen. Ist aber nicht bekannt, welche die langsamste Task ist, kommt ein etwas komplexeres Synchronisierungsverfahren zum Einsatz, von dem ein Programmierbeispiel in **VAL 3** nachstehend angegeben ist.

Beispiel

```
// Synchronization program for N tasks
```

Das nachstehende Programm `synchro(num& n, bool& bSynch, num nN)` muss bei jeder Task zur Synchronisierung aufgerufen werden.

Die Variable `n` muss auf 0, `bSynch` auf `false`, sowie `nN` auf die Anzahl zu synchronisierender Tasks initialisiert werden.

```
begin
  n = n + 1
  // Task synchronization waiting instruction
  // makes sure all tasks are waiting here to resume operation
  wait((n==nN) or (bSynch==true))
  bSynch = true
  n = n - 1
  // Task release waiting instruction
  // makes sure all tasks have resumed operation to clear synch context
  wait((n==0) or (bSynch == false))
  bSynch = false
end
```

5.9. GEMEINSAME NUTZUNG VON RESSOURCEN

Wenn mehrere Tasks gleiche Systemkomponenten oder Elemente der Roboterzelle (globale Variablen, Monitore, Tastatur, Roboter usw.) verwenden, muss darauf geachtet werden, dass sie sich nicht gegenseitig stören.

Zum Schutz der Komponenten kann eine sogenannte "('mutex')"-Funktion (mutuel exclusion) verwendet werden, die sie nur für jeweils eine Task freigibt. Das nachstehende Programmierbeispiel verdeutlicht die Funktion Mutex in **VAL 3**.

Beispiel

Diese Programmanzeige (num c) füllt den Bildschirm mit den gleichen Zeichen und gewährleistet, dass keine andere Task gleichzeitig mit demselben Programm auf dem Bildschirm schreibt. bScreen muss auf **false** initialisiert werden.

```
begin
  // make sure only one task accesses the screen at one time
  setMutex(bScreen)
  c=c%10
  // write something to the screen with chars
  sOutput="Hello world"
  // wait for screen refresh
  delay(0.2)
  // let other tasks access the screen now
  bScreen=false
end
```

5.10. VAL 3 WATCHDOG

Der **VAL 3** WatchDog bietet die Möglichkeit, die Nutzung des Roboters zu sichern. Er bietet die Möglichkeit, sicherzustellen, dass der Roboter nur benutzt werden kann, wenn eine **VAL 3**-Task aktiviert ist.

Das System kann mit oder ohne **VAL 3** WatchDog-Funktion eingestellt werden. Wenn das System mit der **VAL 3** WatchDog-Funktion konfiguriert wird, kann die Stromversorgung des Roboters nur eingeschaltet werden, wenn eine **VAL 3**-Applikation den WatchDog aktiviert lässt. Damit WatchDog aktiviert bleibt, muss die entsprechende **VAL 3** Funktion **wdgRefresh()** aufgerufen werden, bevor die Zeit **val3WatchdogPeriod** abgelaufen ist.

Der WatchDog wird mit dem Parameter **WatchdogPeriod** in der cell.cfx-Konfigurationsdatei konfiguriert:

| | |
|--|---|
| val3WatchdogPeriod<=0 | Das System startet ohne WatchDog. Der Standartwert von val3WatchdogPeriod ist -1 (kein WatchDog). |
| val3WatchdogPeriod>0 (in Sekunden) | Das System startet mit dem aktivierten WatchDog. Die Stromversorgung des Roboters kann nur eingeschaltet werden, wenn der WatchDog gestartet wurde. Wenn der WatchDog aktiviert ist, wird eine Meldung zum Protokoll beim Systemboot hinzugefügt. |

Die **VAL 3**-Funktion **wdgRefresh()** aktiviert ihn:

bool wdgRefresh()

Funktion

Diese Funktion startet und aktualisiert den WatchDog. Sie gibt den aktuellen Status des WatchDog zurück.

| | |
|--------------|--|
| false | Die Funktion hat den WatchDog gestartet (erster Aufruf der Funktion oder nach Ablauf der Zeit WatchDog). |
| true | Der WatchDog ist aktiviert und wird regelmäßig aktualisiert... |

Nach Ablauf der Zeit WatchDog wird der Roboter gestoppt 'cat 1'(**EstopSoft**). Durch erneutes Aufrufen der Funktion **wdgRefresh** wird Estop sofort gelöscht.

Wenn die Funktion **wdgRefresh** mit einem System benutzt wird, das ohne **WatchDog** konfiguriert wurde, wird der **Runtime Error 045** generiert. In diesem Fall ist es nicht mehr möglich, den Roboter einzuschalten (ein Neustart ist erforderlich).

5.11. ANWEISUNGEN

void taskSuspend(string sName)

Funktion

Diese Anweisung unterbricht die Ausführung der Task **sName**.

Wenn die Task bereits den Status **STOPPED** hat, ist die Anweisung wirkungslos.

Falls **sName** nicht einer **VAL 3**-Task entspricht, bzw. einer **VAL 3**-Task entspricht, die von einer anderen Bibliothek erstellt wurde, entsteht ein Laufzeitfehler.

Siehe auch

void taskResume(string sName, num nSkip)

void taskKill(string sName)

void taskResume(string sName, num nSkip)

Funktion

Diese Anweisung setzt die Task **sName** bei der Zeile fort, die **nSkip** Programmzeilen vor oder nach der aktuellen Zeile liegt.

Wenn **nSkip** negativ ist, wird die Ausführung vor der aktuellen Zeile fortgesetzt. Wenn die Task nicht den Status **STOPPED** hat, ist die Anweisung wirkungslos.

Falls **sName** nicht einer **VAL 3**-Task entspricht, bzw. einer **VAL 3**-Task entspricht, die von einer anderen Bibliothek erstellt wurde oder keine Programmzeile an der definierten **nSkip** vorhanden ist, entsteht ein Laufzeitfehler.

Siehe auch

void taskSuspend(string sName)

void taskKill(string sName)

void taskKill(string sName)

Funktion

Diese Anweisung unterbricht und löscht anschließend die Task **sName**. Nach Ausführung dieser Anweisung ist die Task **sName** im System nicht mehr vorhanden.

Falls keine Task **sName** existiert, oder die Task **sName** von einer anderen Bibliothek erstellt wurde, hat die Anweisung keinerlei Auswirkung.

Siehe auch

void taskSuspend(string sName)
void taskCreate string sName, num nPriority, program(...)

void setMutex(bool& bMutex)

Funktion

Diese Anweisung wartet, bis die Variable **bMutex** false ist und setzt sie dann auf true.

Diese Anweisung erfordert die Verwendung einer booleschen Variable als gegenseitigen Ausschlussmechanismus, um die gemeinsam genutzten Ressourcen zu schützen (Siehe Kapitel 5.9).

string help(num nErrorCode)

Funktion

Die Anweisung liefert die Beschreibung für den angegebenen Laufzeitfehlercode **nErrorCode** der Task. Die Beschreibung erfolgt in der aktuellen Controller-Sprache.

Beispiel

Dieses Programm überprüft, ob ein Fehler an der "Roboter"-Task vorliegt und zeigt dem Bediener gegebenenfalls den Fehlercode an.

```
nErrorCode=taskStatus( "robot" )
if (nErrorCode > 1)
    sOutput=help(nErrorCode)
endif
```

num **taskStatus(string sName)**

Funktion

Diese Anweisung liefert den aktuellen Status der Task **sName** oder den Code des Laufzeitfehlers der Task, wenn sich letztere in Fehlerzustand befindet:

| Code | Beschreibung |
|-------------|--|
| -1 | Es existiert in der aktuellen Bibliothek oder Applikation keine Task sName |
| 0 | Die Task sName wurde ohne Laufzeitfehler unterbrochen (Anweisung taskSuspend() oder Debugmodus) |
| 1 | Die von der aktuellen Bibliothek oder Applikation erstellte Task sName läuft |
| 10 | Ungültige numerische Berechnung (Division durch Null). |
| 11 | Ungültige numerische Berechnung (zum Beispiel In(-1)) |
| 20 | Zugriff auf ein Array mit höherem Index als die Arraygröße. |
| 21 | Zugriff auf ein Array mit negativem Index. |
| 29 | Ungültiger Name der Task. Siehe Anweisung taskCreate() . |
| 30 | Der festgelegte Name entspricht keiner Task von VAL 3VAL 3 . |
| 31 | Es existiert bereits eine Task gleichen Namens. Siehe Anweisung taskCreate() . |
| 32 | Es werden nur 2 verschiedene Perioden für die synchronen Tasks unterstützt. Periode ändern. |
| 40 | Der Speicherplatz genügt nicht für die Daten. |
| 41 | Der Ausführungsspeicher ist für die Task unzureichend. Siehe Größe des Ausführungsspeichers. |
| 60 | Maximale Ausführungszeit der Anweisung ist abgelaufen. |
| 61 | Interner Fehler des VAL 3 -Interpreters |
| 70 | Ungültiger Wert des Parameters der Anweisung. Siehe entsprechende Anweisung. |
| 80 | Verwendung von Daten oder Programmen einer nicht im Speicher geladenen Bibliothek. |
| 81 | Die Kinematik ist nicht kompatibel: Verwendung eines point/joint/config , welcher nicht mit der Kinematik des Arms kompatibel ist. |
| 82 | Referenz-Frame oder -Tool einer Variablen gehören zu einer Bibliothek und sind nicht über den Geltungsbereich der Variablen zugänglich (Bibliothek nicht im Projekt der Variablen angegeben oder Referenzvariable ist privat). |
| 90 | Die Task kann an der festgelegten Stelle nicht wieder aufgenommen werden. Siehe Anweisung taskResume() . |
| 100 | Die im Bewegungsdeskriptor spezifizierte Geschwindigkeit ist ungültig (negativ oder zu groß). |
| 101 | Die im Bewegungsdeskriptor spezifizierte Beschleunigung ist ungültig (negativ oder zu groß). |
| 102 | Der im Motiondescriptor angegebene decel-Wert ist ungültig (negativ oder zu groß). |
| 103 | Der im Motiondescriptor angegebene tvel-Wert ist ungültig (negativ oder zu groß). |
| 104 | Der im Motiondescriptor angegebene rvel-Wert ist ungültig (negativ oder zu groß). |
| 105 | Der im Bewegungsdeskriptor spezifizierte Parameter reach ist ungültig (negativ). |
| 106 | Der im Bewegungsdeskriptor spezifizierte Parameter leave ist ungültig (negativ). |
| 122 | Schreibversuch auf einem Eingang des Systems. |
| 123 | Verwendung eines Ein-/Ausgangs dio , aio oder sio , der keinem Ein-/Ausgang des Systems zugeordnet ist. |
| 124 | Versuch des Zugriffs auf einen geschützten Ein-/Ausgang des Systems |
| 125 | Lese- oder Schreibfehler an einem dio , aio oder sio (Fehler auf Feldbus) |
| 150 | Diese Bewegungsanweisung lässt sich nicht ausführen: Eine vorher angeforderte Bewegung konnte nicht beendet werden (Punkt nicht erreichbar, singulärer Punkt, Konfigurationsproblem usw.) |
| 153 | Bewegungsbefehl nicht unterstützt |
| 154 | Ungültige Bewegungsanweisung: Ziel nicht erreichbar oder Bewegungsdeskriptor überprüfen. |
| 160 | Koordinaten des Tools flange nicht gültig |
| 161 | Koordinaten des Koordinatensystems world nicht gültig |
| 162 | Verwendung eines point ohne Referenzsystem. Siehe Definition. |
| 163 | Verwendung eines Koordinatensystems ohne Referenzsystem. Siehe Definition. |
| 164 | Verwendung eines Tools ohne Referenztool. Siehe Definition. |
| 165 | Referenzsystem oder Referenztool ungültig (globale Variable mit lokaler Variablen verbunden) |
| 250 | Es ist keine Runtime-Lizenz für diese Anweisung installiert bzw. die Demo-Lizenz ist nicht mehr gültig. |

Siehe auch

void taskResume(string sName, num nSkip)
void taskKill(string sName)

void taskCreate string sName, num nPriority, program(...)

Funktion

Diese Anweisung erstellt und startet die Task **sName**.

sName muss 1 bis 15 Zeichen zwischen "a..zA..Z0..9_" besitzen. Keine andere von derselben Bibliothek erstellte Task darf denselben Namen haben.

Die Ausführung von **sName** beginnt mit dem Aufruf von **program** mit den spezifischen Parametern. Lokale Variablen können nicht für als per Referenz übertragene Parameter verwendet werden, um zu garantieren, dass die Variable nicht vor Ausführung der Task gelöscht wird.

Die Task endet vorprogrammiert mit der letzten Zeile von **program** bzw. vorher, wenn diese ausdrücklich gelöscht wurde.

nPriority muss zwischen 1 und 100 liegen. Bei jedem sequentiellen Ordnen der Tasks führt das System die in **nPriority** angegebene Zahl von Programmzeilen aus, bzw. weniger, wenn eine sperrende Anweisung angetroffen wird (siehe Kapitel Sequentielles Ordnen).

Wenn das System nicht genügend Speicherplatz zum Erstellen der Task vorfindet, **sName** ungültig oder in der Bibliothek bereits vergeben oder **nPriority** ungültig ist, wird ein Laufzeitfehler erzeugt.

Beispiel

```
// start a new task to read a message
taskCreate "t1", 10, read(sMessage)
// waits for the end of task t1
wait(taskStatus("t1") == -1)
// Use the message
sOutput=sMessage
```

Siehe auch

void taskSuspend(string sName)
void taskKill(string sName)
num taskStatus(string sName)

void taskCreateSync string sName, num nPeriod, bool& bOverrun, program(...)

Funktion

Diese Anweisung erstellt und startet eine synchrone Task.

Die Ausführung der Task startet mit dem Aufruf des angegebenen Programms mit den spezifischen Parametern. Lokale Variablen können nicht für als per Referenz übertragene Parameter verwendet werden, um zu garantieren, dass die Variable nicht vor Ausführung der Task gelöscht wird.

Wenn das System nicht genügend Speicherplatz zum Erstellen der Task vorfindet, oder falls ein bzw. mehrere Parameter nicht gültig sind, wird ein runtime-Fehler erzeugt.

Eine detaillierte Beschreibung der synchronen Tasks (Siehe Kapitel 5.5).

Parameter

| | |
|---------------------------|--|
| string sName | Name der zu erstellenden Task. Er muss aus 1 bis 15 Zeichen aus dem Bereich " <code>_a..zA..Z0..9</code> " bestehen. Zwei zu derselben Applikation oder Bibliothek gehörende Tasks dürfen nicht denselben Namen haben. |
| num nPeriod | Intervall der zu erstellenden Task (s). Der spezifizierte Wert wird auf ein Vielfaches der Systemtaktperiode abgerundet. Jede positive Periode wird unterstützt, doch das System unterstützt nur 2 verschiedene Perioden synchroner Tasks zur gleichen Zeit. Standardgemäß ist die Systemtaktperiode auf 4 ms (0.004s) festgelegt. |
| bool& bOverrun | Boolesche Variable zur Signalisierung eines Zeitüberlaufs. Nur globale Variablen werden unterstützt, um zu garantieren, dass die Variable nicht vor der Task gelöscht wird. |
| program | Name des beim Starten der Task aufzurufenden VAL 3 -Programms, die entsprechenden Parameter werden in Klammern angegeben. |

Beispiel

```
// Create a supervisor task scheduled every 20 ms
taskCreateSync "supervisor", 0.02, bSupervisor, supervisor()
```

void wait(bool bCondition)

Funktion

Diese Anweisung stellt die aktuelle Task solange zurück, bis **bCondition true** wird.

Die Task bleibt **RUNNING** während der Wartezeit. Wenn **bCondition** bei der ersten Überprüfung **true** ist, wird die Ausführung sofort mit der gleichen Task fortgesetzt (kein sequentielles Ordnen der nächsten Task).

Siehe auch

void delay(num nSeconds)

bool watch(bool bCondition, num nSeconds)

void delay(num nSeconds)

Funktion

diese Anweisung schiebt die Ausführung der laufenden Task für **nSeconds** Sekunden auf.

Die Task bleibt **RUNNING** während der Wartezeit. Wenn **nSeconds** negativ oder gleich null ist, beginnt das System sofort mit der Ausführung des nächsten **VAL 3**-Task.

Beispiel

Dieses Programm wird als Schleife ausgeführt, dabei wird darauf geachtet, dass keine unnötigen CPU-Ressourcen verwendet werden:

```
while
bExit==false
sOutput=toString("", clock()* 10))
// let another task perform its operation immediately
delay(0)
endWhile
```

Siehe auch

num clock()

bool watch(bool bCondition, num nSeconds)

num **clock()**

Funktion

Diese Anweisung liefert den aktuellen Wert des systeminternen Taktgebers in Sekunden.

Die Genauigkeit des internen Taktgebers beträgt eine Millisekunde. Er wird beim Start der Steuerung auf **0** gesetzt und ist unabhängig von der Uhrzeit.

Beispiel

Zur Berechnung der Verzögerung zwischen 2 Anweisungen, den Uhrzeitwert vor der ersten Anweisung speichern:

```
nStart=clock()
```

Nach der letzten Anweisung die Verzögerung folgendermaßen berechnen:

```
nDelay = clock() - nStart
```

Siehe auch

void delay(num nSeconds)
bool watch(bool bCondition, num nSeconds)

bool **watch(bool bCondition, num nSeconds)**

Funktion

Dies Anweisung hält die aktuelle Task an, bis **bCondition** gleich **true** ist oder **nSeconds** Sekunden vergangen sind.

Überträgt **true**, wenn die Wartezeit dadurch beendet wird, dass **bCondition** gleich **true** wird. Wird die Wartezeit beendet, weil die Zeit abgelaufen ist, wird **false** übertragen.

Die Task bleibt **RUNNING** während der Wartezeit. Wenn **bCondition** bei der ersten Überprüfung **true** ist, wird die Ausführung sofort mit der gleichen Task fortgesetzt, andernfalls werden die anderen **VAL 3**-Tasks vom System eingeschoben (auch wenn **nSeconds** kleiner oder gleich **0** ist).

Beispiel

Dieses Programm wartet auf ein Signal und zeigt nach 20 s eine Fehlermeldung an.

```
if (watch (diSignal==true, 20)) == false
  sOutput="Error: waiting for Signal"
  wait(diSignal==true)
endif
```

Siehe auch

void delay(num nSeconds)
void wait(bool bCondition)
num clock()

KAPITEL 6

BIBLIOTHEKEN

6.1. DEFINITION

Eine **VAL 3-Bibliothek** ist eine **VAL 3-Applikation**, die Variablen oder Programme beinhaltet, die von einer anderen Applikation oder anderen **VAL 3-Bibliotheken** verwendet werden können.

Als **VAL 3-Applikation** besteht eine **VAL 3-Bibliothek** aus folgenden Elementen:

- einer Gruppe von **Programmen**: den auszuführenden **VAL 3-Anweisungen**,
- einer Gruppe von **globalen Variablen**: den Daten der Bibliothek
- einer Gruppe von **Bibliotheken**: von der Bibliothek benutzte externe Anweisungen und Variablen

Eine in Ausführung befindliche Bibliothek kann zudem enthalten:

- eine Gruppe von Tasks: eigene Programme der in Ausführung befindlichen Bibliothek

Jede Applikation kann als Bibliothek und jede Bibliothek als Applikation verwendet werden, wenn in ihr die Programme **start()** und **stop()** definiert sind.

6.2. SCHNITTSTELLE

Die globalen Programme und Variablen einer Bibliothek sind entweder **public** oder **private**. Nur die als **public** deklarierten globalen Programme und Variablen sind außerhalb der Bibliothek zugänglich. Als **private** deklarierte Programme und globale Variablen können nur von den Programmen der Bibliothek verwendet werden.

Die Gesamtheit der als **public** deklarierten globalen Programme und Variablen einer Bibliothek bilden ihre Schnittstelle: Mehrere Bibliotheken können ein und dieselbe Schnittstelle besitzen, falls ihre als **public** deklarierten Programme und Variablen denselben Namen haben.

Die vom Programm einer Bibliothek geschaffenen Tasks sind immer **privat**, das heißt, sie sind nur von dieser Bibliothek zugänglich.

6.3. KENNUNG DER SCHNITTSTELLE

Um eine Bibliothek verwenden zu können, muss eine Applikation zunächst einen Identifier deklarieren, dieser Bibliothek zuordnen und dann in einem Programm die Bibliothek mithilfe dieses Identifiers in den Speicher laden. Der Identifier ist der Schnittstelle der Bibliothek zugeordnet, nicht der Bibliothek selbst. Zu diesem Identifier kann eine beliebige Bibliothek mit identischer Schnittstelle geladen werden. Es ist deshalb zum Beispiel möglich, eine Bibliothek für jedes mögliche Teil einer Applikation zu definieren und dann nur das beim jeweiligen Zyklus gerade zu bearbeitende Teil zu laden.

6.4. INHALT

Der Inhalt einer Bibliothek ist nicht vorgeschrieben: Sie kann nur Programme, nur Variablen oder beides gleichzeitig enthalten.

Der Zugang zu einer Bibliothek erfolgt durch Aufschreiben des Namens der Kennung, gefolgt von **:** und dem Namen des Programms oder der Daten der Bibliothek, zum Beispiel:

```
// Load the "article_7" library under the "article" identifier
article:libLoad("article_7")
// Display as title the content of the 'sName' variable of the article_7 library
title(article:sName)
// Call the init() program for the current article
call article:init()
```

Wird der Zugriff auf den Inhalt einer Bibliothek gefordert, die noch nicht in den Speicher geladen wurde, so bewirkt dies einen Laufzeitfehler.

6.5. VERSCHLÜSSELUNG

VAL 3 unterstützt verschlüsselte Bibliotheken auf Basis der verbreitet verwendeten Tools zur ZIP-Komprimierung und Verschlüsselung.

Eine verschlüsselte Bibliothek ist eine standardmäßige ZIP-Datei des Inhalts des Bibliothekverzeichnisses (Achtung: erweiterte 128-Bit und 256-Bit AES-Verschlüsselung wird nicht unterstützt). Der Name der ZIP-Datei muss die Erweiterung '.zip' haben und aus weniger als 15 Zeichen (einschließlich Erweiterung) bestehen. Um eine robuste Verschlüsselung zu garantieren, muss das ZIP-Passwort aus mehr als 10 Zeichen bestehen und darf nicht in Wörterbüchern zu finden sein.

Geheimes Passwort, öffentliches Passwort

Der **VAL 3**-Interpreter muss Zugriff auf das geheime ZIP-Passwort haben, um eine verschlüsselte Bibliothek zu laden; dazu wird ein PC-Tool mit **Stäubli Robotics Suite** zur Codierung des geheimen ZIP-Passworts in ein öffentliches **VAL 3**-Passwort geliefert. Das öffentliche **VAL 3**-Passwort macht es möglich, die verschlüsselte Bibliothek in einem **VAL 3**-Programm zu verwenden. Der Inhalt der Bibliothek bleibt geheim, da das ZIP-Passwort nicht vom **VAL 3**-Passwort aus berechnet werden kann.

Projektverschlüsselung

Die Start-Applikation kann auf dem Controller nicht direkt verschlüsselt werden. Eine vollständige Applikation kann folgendermaßen verschlüsselt werden:

- Sein Startprogramm als öffentlich erklären.
- Eine andere Applikation erstellen, die nur die verschlüsselte Applikation als Bibliothek lädt und ihr Startprogramm aufruft.

6.6. IM SPEICHER LADEN UND AUS DEM SPEICHER ENTFERNEN

Beim Öffnen einer **VAL 3**-Applikation werden alle erklärt Bibliotheken untersucht, um die entsprechenden Schnittstellen aufzubauen. Bei diesem Schritt werden die Bibliotheken nicht in den Speicher geladen.

ACHTUNG:

In einem geschlossenen Kreis resultierende Bibliothekszuweisungen werden nicht unterstützt. Wenn Bibliothek A Bibliothek B benutzt, kann Bibliothek B Bibliothek A nicht benutzen.

Beim Laden einer Bibliothek werden deren globale Variablen initialisiert und ihre Programme auf eventuelle Syntaxfehler überprüft. Wenn mehrere Bibliotheks-Bezeichner dieselbe Bibliothek auf der Speicherplatte laden, nutzen sie dieselbe Bibliothek gemeinsam im Speicher. Dann wird die Bibliothek nur einmal geladen und von allen Bezeichnern wiederverwendet. In unten stehendem Beispiel nutzen lib1 und lib2 dieselben gespeicherten Daten.

```
lib1:libLoad("appData")
lib1:sText = "lib1"
lib2:libLoad("appData")
// The change on lib2:sText applies here also to lib1:sText
lib2:sText = "lib2"
```

Eine Bibliothek muss nicht aus dem Speicher entfernt werden. Dies erfolgt automatisch nach dem Beenden der Anwendung oder beim Laden einer neuen Anwendung an deren Stelle.

Wird eine **VAL 3**-Applikation von der **MCP**-Benutzerschnittstelle gestoppt, so wird zunächst das Programm **stop()** ausgeführt und anschließend werden alle Tasks der Applikation und ihrer Bibliotheken, falls solche verbleiben, gelöscht.

Zugriffspfad

Die Anweisungen **libLoad()**, **libSave()** und **libDelete()** verwenden einen Bibliotheks-Zugriffspfad, der in Form einer Zeichenkette festgelegt ist. Ein Zugriffspfad enthält einen Stamm (fakultativ), einen Pfad (fakultativ) und den Namen einer Bibliothek in folgendem Format:

root://Path/Name

Der Stamm legt den Datenträger der Datei fest: "**Floppy**" für Diskette, "**USB0**" für ein mit einem **USB**-Port verbundenes Peripheriegerät (Stick, Diskette), "**Disk**" für die Flash-Disk des Controllers oder den Namen einer auf dem Controller festgelegten **Ftp**-Verbindung für einen Netzzugang.

Im Standardfall ist der Stamm "**Disk**" und der Pfad leer.

Beispiele

```
// load library "article_1" on Disk Equivalent to "Disk://article_1"
article:libLoad("article_1")
// Save library on USB device
article:libSave("USB0://articles/article_1")
// Load the default article defined within the current application
article:libLoad("./defaultArticle")
```

Fehlercodes

Die **VAL 3**-Funktionen für den Umgang mit Bibliotheken erzeugen keine Laufzeitfehler, sondern übergeben einen Fehlercode, mit dessen Hilfe das Ergebnis einer Anweisung überprüft und der Ursprung eventueller Probleme untersucht werden kann.

| Code | Beschreibung |
|----------------|--|
| 0 | Kein Fehler |
| 1 | Die Bibliothek wurde geladen, aber mit einer Warnmeldung. Einzelheiten zur Ursache der Warnmeldung siehe Protokoll. |
| 10 | Die Kennung der Bibliothek wurde nicht durch libLoad() initialisiert. |
| 11 | Bibliothek geladen, aber öffentliche Schnittstelle stimmt nicht überein. Ein Laufzeitfehler 80 entsteht, wenn das VAL 3 -Programm versucht auf fehlende Elemente zuzugreifen. Siehe Anweisungen libExist oder isDefined() . |
| 12 | Laden der Bibliothek unmöglich: Die Bibliothek enthält ungültige Daten oder Programme, oder bei einer verschlüsselten Bibliothek ist das angegebene Benutzerpasswort nicht richtig. |
| 13 | Entfernen der Bibliothek nicht möglich: Die Bibliothek wird von einer anderen Task genutzt. |
| 14 | Entfernen der Bibliothek nicht möglich: In der Bibliothek läuft eine VAL 3 -Task. Alle von VAL 3 -Programmen der Bibliothek erstellten Tasks müssen vor dem Entladen der Bibliothek beendet werden. |
| 20 | Datei-Zugriffsfehler: der Stamm des Pfads ist ungültig. |
| 21 | Datei-Zugriffsfehler: der Pfad ist ungültig. |
| 22 | Datei-Zugriffsfehler: der Name ist ungültig. |
| 23 | Verschlüsselte Bibliothek erwartet. Bibliothek ist nicht oder schlecht verschlüsselt. |
| 24 | Die Bibliothek kann nicht geladen werden, da sie einen Benutzertyp benutzt, der nicht geladen werden kann. |
| >=30 | Lese-/Schreibfehler auf Datei. |
| 31 | Bibliothek kann nicht abgespeichert werden: Der angegebene Pfad enthält bereits eine Bibliothek. Um eine Bibliothek auf der Festplatte zu ersetzen, muss sie erst mit libDelete() gelöscht werden. |
| 32 | Treibermeldung "Gerät nicht gefunden" |
| 33 | Treibermeldung "Gerätefehler" |
| 34 | Treibermeldung "Gerätetimeout" |
| 35 | Treibermeldung "Gerät schreibgeschützt" |
| 36 | Treibermeldung "Keine Platte vorhanden" |
| 37 | Treibermeldung "Platte nicht formatiert" |
| 38 | Treibermeldung "Platte voll" |
| 39 | Treibermeldung "Datei nicht gefunden" |
| 40 | Treibermeldung "Schreibgeschützte Datei" |
| 41 | Treibermeldung "Verbindung verweigert" |
| 42 | Treibermeldung "FTP-Server antwortet nicht" |
| 43 | Treibermeldung "FTP-Kernelfehler" |
| 44 | Treibermeldung "FTP-Parameterfehler" |
| 45 | Treibermeldung "FTP-Zugriffsfehler" |
| 46 | Treibermeldung "FTP-Verzeichnis voll" |
| 47 | Treibermeldung "Ungültiger FTP Nutzer-Login" |
| 48 | Treibermeldung "FTP-Verbindung nicht definiert" |

6.7. ANWEISUNGEN

num identifier:libLoad(string sPath)

num identifier:libLoad(string sPath, string sPassword)

Funktion

Diese Anweisung initialisiert den Bezeichner einer Bibliothek durch Laden der Programme und Variablen der Bibliothek **sPath**. Der angegebene (optionale) Parameter **sPassword** wird als Passwort für verschlüsselte Bibliotheken verwendet. Das angegebene **sPassword** muss das vom geheimen ZIP-Passwort der verschlüsselten Bibliothek berechnete öffentliche **VAL 3**-Passwort sein (siehe Kapitel 6.5, Seite 108).

Diese Anweisung liefert **0** nach erfolgreichem Laden und einen Bibliotheks-Fehlercode, wenn noch von der zu entfernenden Bibliothek erstellte Tasks existieren, wenn der Zugangspfad zur Bibliothek ungültig ist, wenn die Bibliothek Syntax-Fehler enthält oder wenn die genannte Bibliothek nicht der für den Identifier angegebenen Schnittstelle entspricht.

Siehe auch

num identifier:libSave(), **num libSave()**

num identifier:libSave(), **num libSave()**

Funktion

Diese Anweisung speichert die Variablen und Programme, die einem Bibliotheks-Bezeichner zugeordnet sind. Wenn **libSave()** ohne Bezeichner aufgerufen wird, wird die Applikation, die die Anweisung **libSave()** enthält gespeichert. Bei Angabe eines Parameters erfolgt das Sichern unter dem angegebenen **sPathn**. Andernfalls wird über den beim Laden festgelegten Pfad gesichert.

Die Anweisung liefert **0**, wenn die Programme und Daten gesichert worden sind, und einen Fehlercode, wenn der Identifier nicht initialisiert ist, wenn der Pfad ungültig ist, oder wenn ein Schreibfehler aufgetreten ist bzw. der genannte Pfad bereits zu einer Bibliothek führt.

ACHTUNG:

Einige Geräte, wie z.B. die Flash-Disk des Controllers unterstützen nur eine begrenzte Anzahl an Schreibzugriffen. Wird in einem Programm häufig **libSave()** verwendet (einmal pro Minute oder öfter) muss dies über ein Gerät erfolgen, dass dies unterstützt.

Siehe auch

num libDelete(string sPath)

num libDelete(string sPath)

Funktion

Diese Anweisung löscht die durch **sPath** angegebene Bibliothek.

Die Anweisung liefert **0**, wenn die angegebene Bibliothek nicht existiert oder gelöscht wurde und einen Fehlercode, wenn der Bezeichner nicht initialisiert ist, wenn der Pfad ungültig ist, oder wenn ein Schreibfehler aufgetreten ist.

Siehe auch

num identifier:libSave(), **num libSave()**
string identifier:libPath(), **string libPath()**

string identifier:**libPath()**, string **libPath()**

Funktion

Diese Anweisung überträgt den Pfad der mit dem Bezeichner verknüpften Bibliothek, oder der aufrufenden Anwendung, falls kein Bezeichner angegeben ist.

Siehe auch

bool libList(string sPath, string& sContents[])

bool libList(string sPath, string& sContents[])

Funktion

Diese Anweisung schreibt den Inhalt des mit **sPath** angegebenen Pfads in das Array **sContents**. Der Rückgabewert ist **true**, wenn das Array **sContents** das gesamte Ergebnis auflistet und **false** wenn das Array zu klein ist, um die gesamte Liste aufzunehmen.

Alle Elemente des Arrays **sContents** werden zuerst auf den Wert "" (leerer String) initialisiert. Nach Ausführung von **libList()** erhält man demnach das Ende der Liste durch Suchen des ersten leeren Strings im Array **sContents**.

Falls **sContents** eine globale Variable ist, wird das Array automatisch vergrößert, um die Speicherung des vollständigen Ergebnisses zu ermöglichen.

Siehe auch

string identifier:libPath(), string libPath()

bool identifier:libExist(string sSymbolName)****

Funktion

Die Anweisung **libExist** testet, ob ein Symbol (globale Daten oder Programm) in einer Bibliothek definiert ist. Es liefert true, wenn das Symbol existiert und zugänglich (öffentliche) ist, andernfalls false.

Der Symbolname eines Programms muss den Anhang "(" haben: "mySymbol" bezeichnet einen Datennamen, "mySymbol()" dagegen einen Programmnamen.

Die Anweisung **libExist** eignet sich, um zu testen, ob ein Ein-/Ausgang für einen Controller definiert ist; außerdem hilft sie beim Umgang mit der Entwicklung der Schnittstelle einer Bibliothek und bietet Möglichkeiten zur Fallunterscheidung, je nachdem, ob es sich um eine neuere oder ältere Version der Schnittstelle handelt.

Beispiel

Dieses Beispiel testet die Schnittstelle einer Bibliothek.

```
// Load part library
nLoadCode = part:libLoad(sPartPath)
// part:sVersion was not defined in the first version of the library
// Test if this library defines it
if (nLoadCode==0) or (nLoadCode==11)
  if (part:libExist("sVersion")==false)
    // initial version
    sLibVersion = "v1.0"
  else
    sLibVersion = part:sVersion
  endIf
endIf
```

Dieses Programm ruft das eventuell vorhandene 'init'-Programm der 'protocol'-Bibliothek auf:

```
if(protocol:libExist("init()") == true)
  call protocol:init()
endif
```

Siehe auch
bool isDefined(*)

KAPITEL 7

BENUTZERTYP

7.1. DEFINITION

Ein Benutzertyp ist ein innerhalb einer **VAL 3**-Applikation definierter Typ, in der er als standardmäßiger Typ verwendet werden kann. Ein Benutzertyp kombiniert einfache, strukturierte oder sogar andere Benutzertypen in einem neuen Datentyp. Benutzertypen steigern die Abstraktionsstufe von Programmen und machen es einfacher sie zu verstehen, zu entwickeln und zu warten. Sie erfordern jedoch eine größere anfängliche Design-Anstrengung, um die adäquaten Typen zu identifizieren, die den Applikationszwängen am besten entsprechen.

Ein Benutzertyp ist ein Satz von Komponenten, wobei jede Komponente aus Folgendem besteht:

- Namen: eine Zeichenkette
- einen Datentyp (einfacher Typ, strukturierter Typ oder Benutzertyp)
- einen Daten-Container (Element, Array oder Collection)
- den Default-Werten

Die Komponenten der standardmäßigen **VAL 3**-Typen benutzen stets einen Elemente-Container (mit einem einzigen Wert). Die Komponenten von Benutzertypen können Array- oder Collection-Container nutzen und deshalb mehrere Werte enthalten. Der standardmäßige Wert für die Komponente definiert die standardmäßige Anzahl an Elementen im Komponenten-Container und den standardmäßigen Wert für jedes dieser Elemente. In einer mit einem Benutzertyp definierten Variablen können jederzeit nicht nur die Werte der Komponenten sondern auch die Anzahl an Elementen in den Komponenten-Containern geändert werden.

7.2. ERSTELLUNG

Die Komponenten eines Benutzertypen haben die gleichen Eigenschaften wie die globalen Daten einer **VAL 3**-Applikation. Deshalb besteht die Erstellung eines neuen Benutzertypen lediglich darin eine **VAL 3**-Applikation auszuwählen und mit einem Namen zu verbinden.

- Der Satz öffentlicher globaler Daten in der ausgewählten Applikation definiert den Satz an Benutzertypkomponenten mit ihrem standardmäßigen Wert.
- Der Name definiert den für den neuen Benutzertypen in der Applikation zu verwendenden Namen in der er definiert ist.

Die privaten Daten und die zur Typendefinition benutzten Programme der Applikation werden im Benutzertyp nicht berücksichtigt.

Sobald ein neuer Benutzertyp in einer Applikation definiert worden ist, können Daten dieses Typs erstellt werden. Die sich daraus ergebende Applikation kann ebenfalls auch zur Typendefinition verwendet werden.

7.3. EINSATZFALL

Die Komponenten einer Benutzertyp-Variablen sind mit einem '' gefolgt vom Namen der Komponenten zugänglich: userVariable.field1.field2 bezieht sich auf den Wert der 'field2' Komponente der 'field1' Komponente der Daten userVariable. Das Erstellen oder Löschen von Elementen in einem Komponenten-Container werden wie für eine Variable mit den Anweisungen `insert()`, `delete()`, `append()` oder `resize()` unterstützt. Beim Erstellen eines neuen Elements wird jeder Komponente ein standardmäßiger Wert zugewiesen, der aus den Elementen und deren Werten besteht, die in der als Typendefinition benutzten Applikation definiert werden.

ACHTUNG:

Die Komponenten der Typen tool, point und frame haben standardmäßig keine Verlinkung.

Der Operator '=' wird stets zwischen zwei Variablen desselben Benutzertyps definiert. Nachdem der Operator '=' ausgeführt worden ist, ist die linke Variable eine Kopie der rechten Variablen: die Komponenten haben dieselbe Anzahl an Elementen in ihren Containern und die Elemente denselben Wert.

KAPITEL 8

STEUERUNG DES ROBOTERS

In diesem Kapitel sind die Anweisungen für die verschiedenen Komponenten des Roboters aufgeführt.

8.1. ALLGEMEINE ANWEISUNGEN

void disablePower()

Funktion

Diese Anweisung schaltet die Armleistung ab und wartet, bis die Leistung tatsächlich abgeschaltet wurde.

Wenn der Arm zu diesem Zeitpunkt in Bewegung ist, erfolgt vor dem Abschalten ein sofortiger Stillstand auf der Bahn.

Siehe auch

void enablePower()
bool isPowered()

void enablePower()

Funktion

Diese Anweisung schaltet den Arm im ferngesteuerten Betrieb ein.

Diese Anweisung hat keinen Einfluss auf den lokalen, manuellen oder Testbetrieb oder wenn die Leistung gerade abgeschaltet wird. Sie erstellt eine Meldung im Protokoll. Vermeiden Sie deshalb wiederholte unverzögerte Einschaltversuche.

Beispiel

```
// Switches on the power and waits for the arm power to be switched on
enablePower()
if(watch(isPowered(), 5) == false)
    sOutput="Arm power supply cannot be switched on"
endif
```

Siehe auch

void disablePower()
bool isPowered()

bool isPowered()

Funktion

Diese Anweisung liefert den Status der Leistungsversorgung des Arms:

true: Armleistung eingeschaltet

false: die Armleistung ist ausgeschaltet bzw. wird gerade eingeschaltet

bool isCalibrated()

Funktion

Diese Anweisung liefert den Kalibrierstatus des Roboters:

true: Alle Roboterachsen sind kalibriert

false: Mindestens eine Roboterachse ist nicht kalibriert

num workingMode(), num workingMode(num& nStatus)

Funktion

Diese Anweisung liefert die aktuelle Betriebsart des Roboters:

| Modus | Status | Betriebsart | Status |
|--------------|---------------|---|--|
| 0 | 0 | Ungültig oder Betriebsartenwechsel | - |
| 1 | 0 | Handbetrieb | Programmierte Bewegung |
| | 1 | | Anschlussbewegung |
| | 2 | | Joint jogging |
| | 3 | | Kartesisch (Frame jogging) |
| | 4 | | Tool jogging |
| | 5 | | zum Punkt (Point jogging) |
| | 6 | | Hold |
| 2 | 0 | Test | programmierte Bewegung (< 250 mm/s) |
| | 1 | | Anschlussbewegung (< 250 mm/s) |
| | 2 | | programmierte schnelle Bewegung (> 250 mm/s) |
| | 3 | | Hold |
| 3 | 0 | lokal | Move (programmierte Bewegung) |
| | 1 | | Move (Anschlussbewegung) |
| | 2 | | Hold |
| 4 | 0 | ferngesteuert | Move (programmierte Bewegung) |
| | 1 | | Move (Anschlussbewegung) |
| | 2 | | Hold |

num esStatus()

Funktion

Diese Anweisung liefert den Status des Not-Aus-Kreises:

| Code | Status |
|-------------|--|
| 0 | Keine Sicherheitsabschaltung. |
| 1 | Warten auf sicheren Neustart. |
| 2 | SS1 - Bedingung für Sicherheitsabschaltung. |
| 3 | SS2 - Abschaltbedingungen für sicheren Betrieb. |

Siehe auch

num workingMode(), num workingMode(num& nStatus)

num getMonitorSpeed()

Funktion

Diese Anweisung liefert die aktuelle Monitorgeschwindigkeit (im Bereich [0, 100]) des Controllers zurück.

Beispiel

Dieses Programm muss in einem Task aufgerufen werden und prüft, dass der erste Roboterzyklus mit niedriger Geschwindigkeit ausgeführt wird:

```
while true
  if(nCycle < 2)
    if (getMonitorSpeed()> 10)
      stopMove()
      sOutput="For the first cycle the monitor speed must remain at 10%"
      wait(getMonitorSpeed( )<= 10)
    endif
    restartMove()
  endif
  delay(0)
endWhile
```

Siehe auch

[num setMonitorSpeed\(num nSpeed\)](#)

num setMonitorSpeed(num nSpeed)

Funktion

Diese Anweisung ändert die aktuelle Monitorgeschwindigkeit des Roboters. [setMonitorSpeed\(\)](#) ist stets in der Lage, die Monitorgeschwindigkeit zu reduzieren. Um sie zu erhöhen, ist [setMonitorSpeed\(\)](#) nur effektiv wenn sich der Roboter in der Betriebsart ferngesteuert Automatik befindet und der Bediener keinen Zugriff auf die Monitorgeschwindigkeit hat (wenn das aktuelle Benutzerprofil die Nutzung der Geschwindigkeitstasten nicht gestattet oder das MCP abgesteckt wurde).

Überträgt 0, wenn die Monitorgeschwindigkeit erfolgreich geändert wurde, ansonsten einen negativen Fehlercode:

| Code | Beschreibung |
|------|--|
| -1 | Der Roboter befindet sich nicht in der Betriebsart ferngesteuert Automatik |
| -2 | Die Monitorgeschwindigkeit wird vom Bediener kontrolliert: aktuelles Benutzerprofil ändern, um den Bedienerzugriff auf die Monitorgeschwindigkeit zu unterbinden |
| -3 | Die angegebene Geschwindigkeit wird nicht unterstützt: Sie muss im Bereich [0, 100] liegen |

Siehe auch

[num getMonitorSpeed\(\)](#)

string getVersion(string sComponent)

Funktion

Diese Anweisung gibt die Version verschiedener Hardware- und Software-Komponenten des Roboter-Controllers zurück. Die unten stehende Tabelle zeigt die Liste der unterstützten Komponenten und für jede von ihnen das Format des zurückgegebenen Werts:

| Komponente | Beschreibung |
|-------------------------|--|
| "System" | Version des Steuersystems, wie "s8.0cs9_B587" |
| "System Date" | Datum des Steuersystems, wie "Jun 1 2016" |
| "System Time" | Datum des Steuersystems, wie "11:59:29" |
| "Arm Type" | Typ des mit dem Controller verbundenen Arms, wie z.B. "tx2_90-S1" oder "ts60-S1-D20- L200" |
| "Arm Tuning" | Version des Armtunings, wie z.B. "R3" |
| "Arm Mounting" | Montagetyp des Arms wie z.B. "floor" (Boden), "wall" (Wand) oder "ceiling" (Decke) |
| "Arm S/N" | Seriennummer des Arms, wie z.B. "F07_12R3A1_A_01" |
| "Controller S/N" | Seriennummer des Controller, wie z.B. "F07_12R3A1_C_01" |
| "Configuration Version" | Version der Steuerungskonfigurationsdatei, wie "c2.009" |
| Lizenzname | Status der Software-Lizenz des Controllers: <ul style="list-style-type: none"> "" (leere Zeichenkette): nicht installiert oder Frist für Demo-Version abgelaufen "demo": Lizenz für eine begrenzte Zeit aktiviert "enabled": Lizenz aktiviert Die Namen der installierten Steuerungslizenzen (wie "alter", "remoteMcp", "oemLicense", "testMode", "mcpMode", ...) sind in der Systemsteuerung des Roboter-Handbediengeräts aufgeführt |

Beispiel

```
if getVersion( "oemlicense" ) != "Enabled"
  sOutput="The compliance license is missing on the controller"
endif
```

Siehe auch

string getLicence(string sOemLicenceName, string sOemPassword)

joint getJntRef(string sReferenceName)

Funktion

Diese Funktion gibt den Wert der Gelenkartikelnummer mit der Bezeichnung sReferenceName zurück. Wenn sReferenceName nicht einer bereits vorhandenen Artikelnummer entspricht, wird ein Laufzeitfehler generiert.

Beispiel

```
jsafeRef1 = getJntRef( "safeReference1" )
jsafeRef2 = getJntRef( "safeReference2" )
```

8.2. STROMEINSPARANWEISUNGEN

Es ist möglich, den Stromverbrauch des Roboters zu reduzieren, wenn sich der Roboter nicht bewegt, indem die Stromversorgung der Antriebs- und Wegmesssystemschnittstellen abgeschaltet wird. Dies muss über die Applikation unter Berücksichtigung der folgenden Anweisungen erfolgen.

void hibernateRobot()

Funktion

Diese Funktion schaltet die Stromversorgung der Antriebs- und Wegmesssystemschnittstellen ab.

| Code | |
|-----------|---|
| 0 | Kein Fehler. |
| -1 | Funktion nicht unterstützt. |
| -2 | Timeout für Warten auf Bestätigung der Sicherheitskarte. |
| -3 | Roboter muss deaktiviert werden, um den Stromsparmodus zu aktivieren. |
| -4 | Timeout bis Antriebs- und Wegmesssystemschnittstelle abgeschaltet sind. In dieser Situation muss die Funktion powerSaveOff zur Wiederherstellung aufgerufen werden. |

Siehe auch

void wakeUpRobot()

void disablePower()

void wakeUpRobot()

Funktion

Diese Funktion bringt den Roboter wieder in den betriebsbereiten Zustand.

| Code | |
|-----------|---|
| 0 | Kein Fehler. Diese Funktion gibt auch 0 zurück, wenn hibernateRobot() nicht vorher aufgerufen wurde. |
| -1 | Funktion nicht unterstützt. |
| -2 | Timeout bis Wegmesssystemschnittstelle betriebsbereit ist. |
| -3 | Timeout bis Sicherheitskarte betriebsbereit ist. |

Siehe auch

void hibernateRobot()

8.3. ANWEISUNGEN ZUR VERWALTUNG VON STROMAUSFÄLLEN

Der J213-Steckverbinder des Computereinschubs (CPT) ermöglicht die Nutzung einer externen 24 V-Stromversorgung. Durch die Verwendung einer sicheren Stromversorgung kann die Applikation das Verhalten bei Stromausfall verwalten (insbesondere Sicherung der Applikationsdaten).

Standardgemäß schließt die Steuerung, wenn ein Stromausfall festgestellt wird (powerSupplyIO/mainPowerOk-Eingang), alle Dateien und sperrt den Zugriff auf das Dateisystem. Dieses Verhalten kann durch Einstellung des Flag **cpuExtPowerSupply** auf True in der Datei /usr/configs/controller.cfx geändert werden.

Wenn das Flag **cpuExtPowerSupply** auf True gesetzt ist, reagiert die Steuerung bei einem Stromausfall nicht. Daher ist die VAL 3-Applikation für die Überwachung des Eingangs powerSupplyIO/mainPowerOk verantwortlich.

Wenn die VAL 3-Applikation einen Stromausfall feststellt (powerSupplyIO/mainPowerOk ist 0), müssen alle Daten gespeichert werden.

Nach dem Speichern der Daten kann die Applikation entweder: Die Funktion **prepareCpuShutdown()** aufrufen und warten, bis 24 V verschwindet oder bis powerSupplyIO/mainPowerOk wieder auf eins steht und dann die Funktion **wakeUpRobot()** aufrufen, damit der Roboter wieder betriebsbereit ist.

[**bool hasCpuExtPowerSupply\(\)**](#)

Funktion

Diese Funktion ermöglicht das Lesen des aktuellen Verhaltens. Sie gibt folgende Werte zurück:

true: Wenn die Steuerung die Stromversorgung prüft und auf deren Ausfall reagiert (Standardverhalten).

false: Wenn die Steuerung die Stromversorgung nicht prüft und auf deren Ausfall nicht reagiert.

Siehe auch

[**void prepareCpuShutdown\(\)**](#)

[**void prepareCpuShutdown\(\)**](#)

Funktion

Diese Funktion muss von der **VAL 3**-Applikation aufgerufen werden, bevor die externe Stromversorgung des Computereinschubs ausgeschaltet wird. Sie führt das Standardverhalten aus: Schließt alle Dateien und sperrt den Zugang zum Dateisystem.

Siehe auch

[**bool hasCpuExtPowerSupply\(\)**](#)

8.4. BREMTESTANWEISUNG

bool brakeTest(**num**& nBrakeStatus)

Funktion

Diese Funktion führt ein Bremstestverfahren durch. Eine detaillierte Beschreibung des Verfahrens finden Sie im Sicherheitshandbuch.

Die Rückgabewerte spiegeln das Ergebnis des gesamten Verfahrens wider:

| Code | Beschreibung |
|-----------------------|---|
| 0 | Alle Bremsen wurden getestet und sind ok. |
| 1 | Mindestens eine der Bremsen hat den Test nicht erfolgreich bestanden. Sie muss ausgewechselt werden. |
| Negative Werte | Bei mindestens einem der Bremstests wurde ein Fehler festgestellt (das Testverfahren ist misslungen). Siehe nachstehende Fehlercodes. |

Der Array nBrakeStatus kann zum Erhalt des Teststatus für jede Bremse benutzt werden.

nBrakeStatus[0] entspricht der Bremse der Achse 1,

nBrakeStatus[1] entspricht der Bremse der Achse 2,

...

| Code | Beschreibung |
|-----------------------|--|
| 0 | Die Bremse wurde getestet und ist ok. |
| 1 | Der Test wurde nicht ausgeführt, da er nicht angefordert worden ist (für zukünftige Benutzung reserviert). |
| 2 | Die Bremse hat den Test nicht erfolgreich bestanden. Sie muss ausgewechselt werden. |
| Negative Werte | Das Testverfahren ist misslungen. Siehe nachstehende Fehlercodes. |

Fehlercodes:

| Code | Beschreibung |
|-------------|---|
| -1 | Falscher Arbeitsmodus. Arbeitsmodus muss automatisch oder ferngesteuert sein. |
| -2 | Der Roboter bewegt sich. Der Roboter muss gestoppt werden, damit das Bremstestverfahren durchgeführt werden kann. |
| -3 | Der Roboter befindet sich nicht in der brakeTest-Position. |
| -5 | Der Roboter befindet sich nicht in der Betriebsart Deaktiviert. |
| -13 | brakeTest-Verfahren vom Benutzer abgebrochen (killTask). |
| -14 | brakeTest-Daten fehlen in der Konfigurationdatei (system.zfx). |

KAPITEL 9

ARMPOSITION

9.1. EINLEITUNG

In diesem Kapitel sind die verschiedenen Typen von **VAL 3**-Variablen beschrieben, mit denen die in einer **VAL 3**-Applikation verfügbaren Armpositionen programmiert werden können.

Zwei Typen von Positionen sind in **VAL 3** definiert: Achswinkel bezogene Punkte (Typ **joint**), die die Winkelstellung jeder Drehachse beschreiben bzw. die lineare Position jeder linearen Achse sowie kartesische Punkte (Typ **point**), mit denen die kartesische Position des TCP's am Ende des Arms bezüglich eines Frame angegeben wird.

Der Typ **tool** beschreibt ein Tool und seine Geometrie zur Positionierung und Geschwindigkeitssteuerung des Arms, außerdem beschreibt er wie das Tool betätigt wird (digitaler Ausgang, Wartezeit).

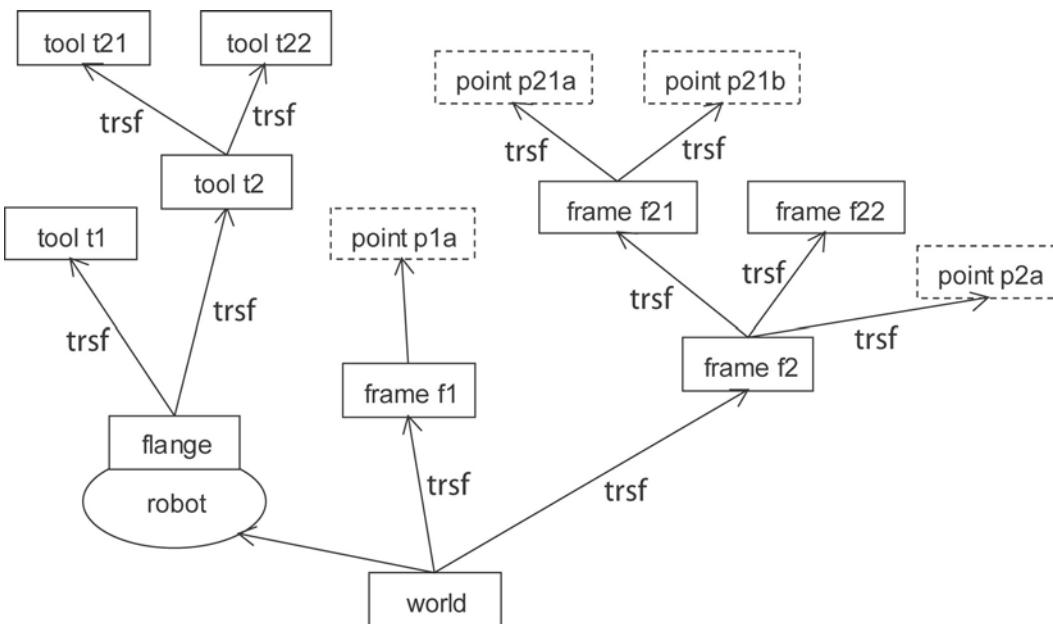
Der Typ **frame** beschreibt ein Bezugs-Koordinatensystem. Durch die Verwendung von Bezugssystemen wird insbesondere die Programmierung bewegter Punkte einfacher und intuitiver.

Der Typ **trsrf** beschreibt eine Koordinatentransformation. Sie wird von den Typen **tool**, **point** und **frame** verwendet.

Der Typ **config** beschreibt den komplexeren Begriff der Armkonfiguration.

Die Beziehungen zwischen den verschiedenen Typen lassen sich wie folgt zusammenfassen:

Beziehung zwischen: frame / point / tool / trsrf



9.2. TYP JOINT

9.2.1. DEFINITION

Ein Achswinkel bezogener Punkt (Typ **joint**) definiert die Winkelposition jeder Drehachse und die lineare Position jeder linearen Achse.

Der Typ **joint** ist ein strukturierter Typ mit folgenden Feldern in der nachstehenden Reihenfolge:

| | |
|-----------------|--|
| num j1 | Joint Position der Achse 1 |
| num j2 | Joint Position der Achse 2 |
| num j3 | Joint Position der Achse 3 |
| num j... | Joint Position der Achse ... (ein Feld pro Achse) |

Diese Felder werden für die Drehachsen in Grad angegeben sowie in Millimetern oder Inches für die linearen Achsen. Der Nullpunkt jeder Achse ist durch den verwendeten Armtyp festgelegt.

Standardmäßig wird der Wert für jedes Feld einer Variable des Typs **joint** mit dem Wert **0** initialisiert.

9.2.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|---|---|
| joint <joint& jPosition1> = <joint jPosition2> | Ordnet jPosition2 der Variablen jPosition1 Feld für Feld zu und überträgt jPosition2 . |
| bool <joint jPosition1> != <joint jPosition2> | Überträgt true , wenn ein Feld von jPosition1 im Rahmen der Robotergenauigkeit nicht gleich dem entsprechenden Feld von jPosition2 ist, ansonsten false . |
| bool <joint jPosition1> == <joint jPosition2> | Überträgt true , wenn jedes Feld von jPosition1 im Rahmen der Robotergenauigkeit gleich dem entsprechenden Feld von jPosition2 ist, ansonsten false . |
| bool <joint jPosition1> > <joint jPosition2> | Überträgt true , wenn jedes Feld von jPosition1 größer als das entsprechende Feld in jPosition2 ist, ansonsten false . |
| bool <joint jPosition1> < <joint jPosition2> | Überträgt true , wenn jedes Feld von jPosition1 kleiner als das entsprechende Feld in jPosition2 ist, ansonsten false . Achtung: jPosition1 > jPosition2 entspricht nicht !(jPosition1 < jPosition2) |
| joint <joint jPosition1> - <joint jPosition2> | Überträgt die Differenz für jedes Feld von jPosition1 mit jPosition2 . |
| joint <joint jPosition1> + <joint jPosition2> | Überträgt die Summe für jedes Feld von jPosition1 und jPosition2 . |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig.

9.2.3. ANWEISUNGEN

joint abs(joint jPosition)

Funktion

Diese Anweisung liefert den Absolutwert jeder Komponente einer Gelenkposition **jPosition**.

Details

Der Absolutwert einer Gelenkposition (Joint) mit den Joint-Operatoren "**>**" oder "**<**" ermöglicht die einfache Berechnung des Abstands zwischen Joint und Referenzposition.

Beispiel

```
jReference = {90, 45, 45, 0, 30, 0}
jMaxDistance = {5, 5, 5, 5, 5, 5}
j = herej()
// Checks that all the axis are less than 5 degrees from the reference
if(!(abs(j - jReference) < jMaxDistance))
    sOutput="Move closer to the marks"
endif
```

Siehe auch

Operator < (joint)
Operator > (joint)

joint herej()

Funktion

Diese Anweisung liefert die aktuelle Gelenkposition des Arms.

Wenn die Armleistung eingeschaltet ist, entspricht der Rückgabewert der vom Controller an die Verstärker gesendeten Position und nicht der am Wegmesssystem abgelesenen Position.

Wenn die Armleistung ausgeschaltet ist, entspricht der Rückgabewert der am Wegmesssystem abgelesenen Position, durch Rauschen bei der Messwertaufnahme kann die Position auch bei angehaltenem Arm geringfügig schwanken.

Die Gelenkposition des Controllers wird alle 4 ms aktualisiert.

Beispiel

```
//Wait until the arm is near the reference position, with a 60 s time out
bStart = watch(abs(herej() - jReference) < jMaxDistance, 60)
if bStart==false
    sOutput="Move closer to the start position"
endif
```

Siehe auch

point here(tool tTool, frame fReference)
bool getLatch(joint& jPosition)
bool isInRange(joint jPosition)

bool isInRange(joint jPosition)

Funktion

Diese Anweisung testet, ob sich die Gelenkposition innerhalb der Software-Grenzen des Arms befindet.

Befindet sich der Arm außerhalb der Software-Grenzwinkel (nach einem Wartungsvorgang), kann der Arm nicht mehr von einer **VAL 3**-Applikation bewegt werden, nur manuelle Bewegungen sind möglich (die Bewegungsrichtung bleibt dabei auf Bewegungen in Richtung der Grenzen beschränkt).

Beispiel

```
// Check if the current position is within the joint limits
if isInRange(herej())==false
    sOutput="Please place the arm within its workspace"
endif
```

Siehe auch

joint herej()

void setLatch(dio dilnput)

Funktion

Diese Anweisung aktiviert die Verriegelung der Position des Roboters für die nächste steigende Flanke des Eingangssignals.

Details

Die Verriegelung der Roboterposition ist eine elektronische Funktion, die nur von den schnellen Eingängen des -Controllers (fln0, fln1) unterstützt wird.

Die Erkennung der steigenden Flanke des Eingangssignals ist nur dann garantiert, wenn das Signal mindestens 31.25 µs lang vor der steigenden Flanke schwach und mindestens 31.25 µs lang nach der steigenden Flanke stark bleibt.

ACHTUNG:

Die Verriegelung wird erst nach einem bestimmten Zeitraum (zwischen 0 und 0.2 ms) nach Ausführung der Anweisung setLatch aktiviert. Sie können nach setLatch die Anweisung delay(0) einfügen, um sich zu versichern, dass die Verriegelung vor Ausführung der nächsten VAL 3-Anweisung aktiviert ist.

Wenn der angegebene digitale Eingang die Verriegelung der Roboterposition nicht unterstützt, wird die Fehlermeldung 70 (ungültiger Parameterwert) erzeugt.

Siehe auch

bool getLatch(joint& jPosition)

bool getLatch(joint& jPosition)

Funktion

Diese Anweisung liest die letzte verriegelte Position des Roboters aus.

Falls eine gültige verriegelte Position gelesen werden kann, sendet die Funktion die Meldung true. Befindet sich eine Verriegelung im Schwebezustand oder wurde sie nie aktiviert, sendet die Funktion die Meldung false und die Position wird nicht aktualisiert.

getLatch gibt so lange die gleiche verriegelte Position an, bis eine neue Verriegelung durch die Anweisung setLatch aktiviert wurde.

Die Position des Roboterarms wird im CS8C-Controller alle 62.5 µs aktualisiert; die verriegelte Position entspricht der Armposition zwischen 0 und 62.5 µs nach der steigenden Flanke des schnellen Eingangs.

Beispiel

```
setLatch(dilatch)
// Wait for setLatch to be effective before using getLatch
delay(0)
// Wait for a latched position during 5 seconds.
bLatch = watch(getLatch(jPosition)==true, 5)
if bLatch==true
    sOutput="Successful position latch"
else
    sOutput="No latch signal was detected"
endif
```

Siehe auch

void setLatch(dio dilnput)
joint herej()

9.3. TYP TRSF

9.3.1. DEFINITION

Eine Transformation (Typ **trsf**) definiert einen Positions- und oder Ausrichtungswechsel. Es ist eine mathematische Zusammensetzung aus einer Translation und einer Drehung.

Eine Transformation selbst stellt keine Position im Raum dar, kann aber als die Position und die Orientierung eines kartesischen Punkts oder Koordinatensystems in Bezug auf ein anderes Koordinatensystem ausgelegt werden.

Der Typ **trsf** ist ein strukturierter Typ mit folgenden Feldern in der nachstehenden Reihenfolge:

| | |
|---------------|---|
| num x | Translation in Richtung der x -Achse |
| num y | Translation in Richtung der y -Achse |
| num z | Translation in Richtung der z -Achse |
| num rx | Rotation um die x -Achse |
| num ry | Rotation um die y -Achse |
| num rz | Rotation um die z -Achse |

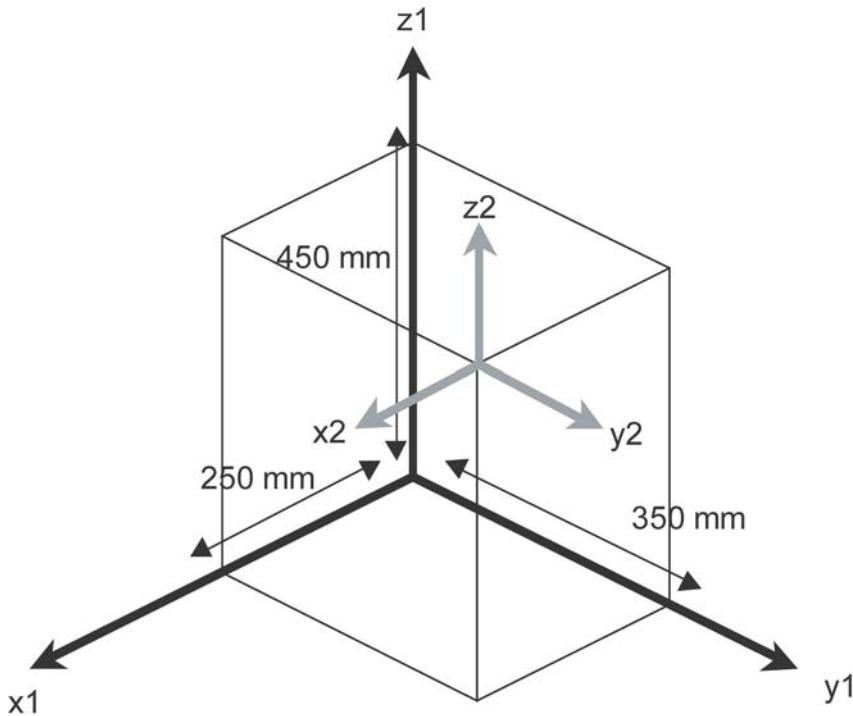
Die Felder **x**, **y** und **z** sind in der Längeneinheit der Applikation (mm oder inch, siehe Kapitel Längeneinheit) angegeben. Die Felder **rx**, **ry** und **rz** sind in Grad angegeben.

Die Koordinaten **x**, **y** und **z** sind die kartesischen Koordinaten der Translation (oder der Position eines Punkts oder Koordinatensystems im Referenzsystem). Wenn **rx**, **ry** und **rz** null sind, ist die Transformation eine Translation ohne Richtungswechsel.

Der vorprogrammierte Wert für eine Variable des Typs **trsf** ist immer **{0,0,0,0,0,0}**.

9.3.2. ORIENTIERUNG

Orientierung



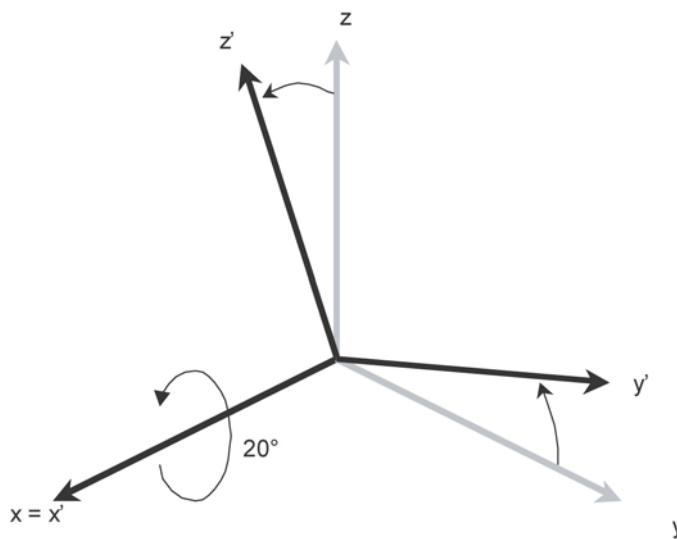
Die Position des Koordinatensystems **R2** (grau) in Bezug auf **R1** (schwarz) ist Folgende:

$$x = 250 \text{ mm}, y = 350 \text{ mm}, z = 450 \text{ mm}, rx = 0^\circ, ry = 0^\circ, rz = 0^\circ$$

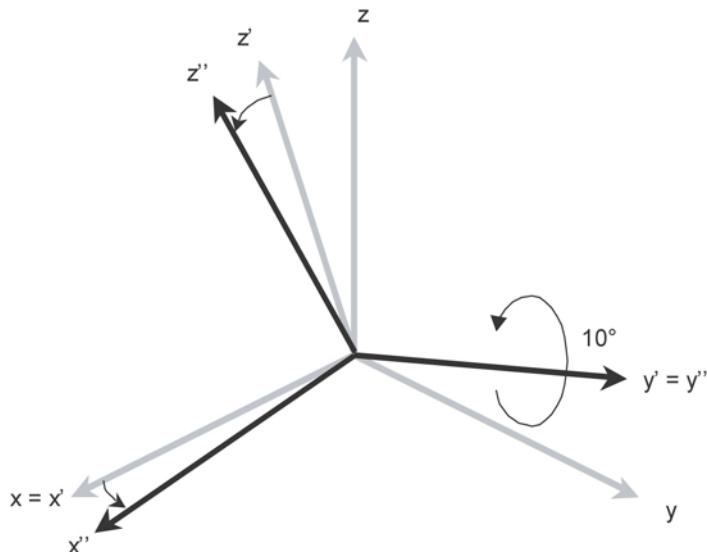
Die drei Koordinaten **rx**, **ry** und **rz** entsprechen den Winkeln, um welche die **x**, **y** und **z**-Achse nacheinander zu drehen sind, um die gewünschte Orientierung des Koordinatensystems zu erhalten.

Die Orientierung **rx = 20°**, **ry = 10°**, **rz = 30°** wird zum Beispiel auf folgende Weise erhalten. Zuerst wird das Koordinatensystem (**x,y,z**) um **20°** um die **x**-Achse gedreht. Damit erhält man das neue Koordinatensystem (**x',y',z'**). Die Achsen **x** und **x'** sind deckungsgleich.

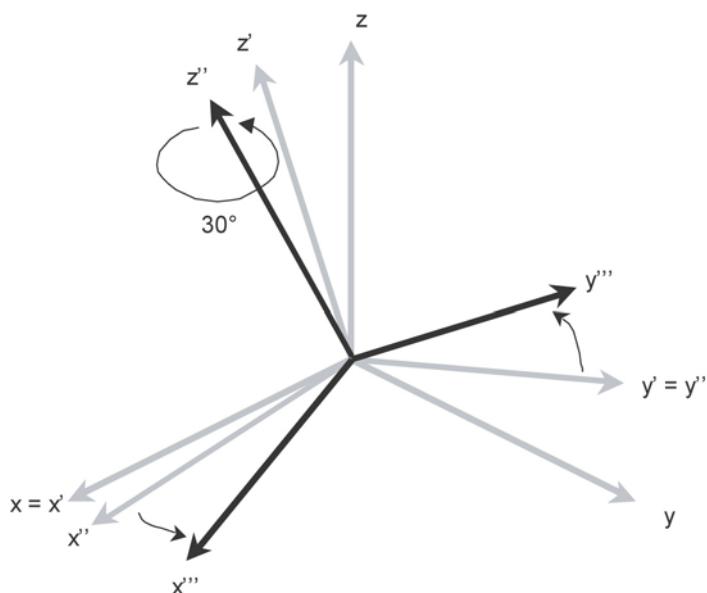
Drehung des Koordinatensystems bezüglich folgender Achse: X



Anschließend wird das Koordinatensystem um **20°** um die Achse **y'** des im vorherigen Schritt erhaltenen Koordinatensystems gedreht. Damit erhält man das neue Koordinatensystem (**x'',y'',z''**). Die Achsen **y'** und **y''** sind deckungsgleich.

Drehung des Koordinatensystems bezüglich folgender Achse: Y'

Zum Schluss wird das Koordinatensystem um **20°** um die Achse **z''** des im vorherigen Schritt erhaltenen Koordinatensystems gedreht. Das neue Koordinatensystem (x''', y''', z''') hat die durch **rx , ry , rz** definierte Orientierung. Die Achsen **z''** und **z'''** sind deckungsgleich.

Drehung des Koordinatensystems bezüglich folgender Achse: Z''

Die Position des Koordinatensystems **R2** (grau) in Bezug auf **R1** (schwarz) ist Folgende:
 $x = 250\text{mm}$, $y = 350 \text{ mm}$, $z = 450\text{mm}$, $rx = 20^\circ$, $ry = 10^\circ$, $rz = 30^\circ$

Die Werte **rx , ry und rz** sind Modulo **360** Grad definiert. Die vom System berechneten Werte für **rx , ry und rz** liegen damit immer zwischen **-180** und **+180**. Ist endweder missverständlich oder falsch: An einem Punkt die Werte RX für 179°, RY für 1° und RZ für 110° eingeben und dann das Fenster schließen und wieder öffnen; daraufhin müssen die Werte gemäß der Beschreibung im Handbuch geändert worden sein.

9.3.3. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|---|
| trsf <trsf& trPosition1> = <trsf trPosition2> | Ordnet trPosition2 der Variablen trPosition1 Feld für Feld zu und überträgt trPosition2 . |
| bool <trsf trPosition1> != <trsf trPosition2> | Überträgt true , wenn ein Feld von trPosition1 nicht gleich dem entsprechenden Feld in trPosition2 ist, ansonsten false . |
| bool <trsf trPosition1> == <trsf trPosition2> | Überträgt true , wenn jedes Feld von trPosition1 gleich dem entsprechenden Feld in trPosition2 ist, ansonsten false . |
| trsf <trsf trPosition1> * <trsf trPosition2> | Überträgt die geometrischen Komponenten der Transformationen trPosition1 und trPosition2 . Achtung! Im Allgemeinen ist trPosition1 * trPosition2 != trPosition2 * trPosition1! |
| trsf ! <trsf trPosition> | Überträgt die Umkehrung der Transformation von trPosition . |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig.

9.3.4. ANWEISUNGEN

num distance(trsf trPosition1, trsf trPosition2)

Funktion

Überträgt die Entfernung zwischen **trPosition1** und **trPosition2**.

ACHTUNG:
Damit die Entfernung gültig ist, müssen Position1 und Position2 im gleichen Referenz-Koordinatensystem definiert sein.

Beispiel

Diese Programmzeile berechnet den Abstand zwischen zwei Tools:

```
distance(position(tTool1, flange), position(tTool2, flange))
```

Siehe auch

point appro(point pPosition, trsf trTransformation)
point compose(point pPosition, frame fReference, trsf trTransformation)
trsf position(point pPosition, frame fReference)
num distance(point pPosition1, point pPosition2)

trsf interpolateL(trsf trStart, trsf trEnd, num nPosition)

Funktion

Diese Anweisung gibt eine Position zwischen der Startposition **trStart** und der Zielposition **trEnd** zurück. Der Parameter **nPosition** spezifiziert die gemäß der Gleichung für die x-Koordinate anzuwendende lineare Interpolation: $\text{trsf.x0} = \text{trStart.x} + (\text{trEnd.x}-\text{trStart.x}) * \text{nPosition}$. Dieselbe Gleichung ist für die Y- und die Z-Koordinaten maßgeblich.

Die Ausrichtung rx, ry, rz wird mit einer ähnlichen, aber komplexeren Gleichung berechnet. Für **interpolateL** wird derselbe Algorithmus verwendet, den der Bewegungsgenerator zur Berechnung von Zwischenpositionen bei einer **movel**-Anweisung nutzt.

interpolateL(trStart, trEnd, 0) liefert **trStart**; **interpolateL(trStart, trEnd, 1)** liefert **trEnd**; **interpolateL(trStart, trEnd, 0.5)** überträgt die mittlere Position zwischen **trStart** und **trEnd**. Ein negativer Wert des Parameters **nPosition** führt zu einer Position 'vor' **trStart**. Ein größerer Wert als 1 führt zu einer Position 'nach' **trEnd**.

Liegt der Parameter **nPosition** nicht im Bereich]-1, 2[wird ein Laufzeitfehler generiert.

Siehe auch

trsf position(point pPosition, frame fReference)
trsf position(frame fFrame, frame fReference)
trsf position(tool tTool, tool tReference)
trsf interpolateC(trsf trStart, trsf trIntermediate, trsf trEnd, num nPosition)
trsf align(trsf trPosition, trsf Reference)

trsf interpolateC(trsf trStart, trsf trIntermediate, trsf trEnd, num nPosition)

Funktion

Diese Anweisung gibt eine durch die Positionen **trStart**, **trIntermediate** und **trEnd** definierte Zwischenposition auf dem Kreisbogen an. Im Parameter **nPosition** wird die anzuwendende Kreisinterpolation spezifiziert. Für **interpolateC** wird derselbe Algorithmus verwendet, den der Bewegungsgenerator zur Berechnung von Zwischenpositionen bei einer **movec**-Anweisung nimmt.

interpolateC(trStart, trIntermediate, trEnd, 0) liefert **trStart**; **interpolateC(trStart, trIntermediate, trEnd, 1)** liefert **trEnd**; **interpolateC(trStart, trIntermediate, trEnd, 0.5)** überträgt die mittlere Position auf dem Bogen zwischen **trStart** und **trEnd**. Ein negativer Wert des Parameters **nPosition** führt zu einer Position 'vor' **trStart**. Ein größerer Wert als 1 führt zu einer Position 'nach' **trEnd**.

Ein Laufzeitfehler wird erzeugt, wenn der Bogen nicht korrekt definiert ist (Positionen zu nahe beieinander) oder wenn die Interpolation der Rotation unbestimmt bleibt (siehe Kapitel "Bewegungskontrolle - Ausrichtungsinterpolation").

Siehe auch

trsf position(point pPosition, frame fReference)
trsf position(frame fFrame, frame fReference)
trsf position(tool tTool, tool tReference)
trsf interpolateL(trsf trStart, trsf trEnd, num nPosition)
trsf align(trsf trPosition, trsf Reference)

trsf align(trsf trPosition, trsf Reference)

Funktion

Diese Anweisung gibt die Eingabe **trPosition** mit geänderter Ausrichtung zurück, so dass die Z-Achse des Ergebnisses auf die nächstliegende X, Y oder Z Achse der Referenzorientierung von **trReference** ausgerichtet wird. Die X, Y, Z Koordinaten von **trPosition** und **trReference** werden nicht verwendet: die x, y, z Koordinaten des Ergebnisses entsprechen den x, y, z Koordinaten von **trPosition**.

Siehe auch

trsf position(point pPosition, frame fReference)
trsf position(frame fFrame, frame fReference)
trsf position(tool tTool, tool tReference)
trsf interpolateL(trsf trStart, trsf trEnd, num nPosition)
trsf interpolateC(trsf trStart, trsf trIntermediate, trsf trEnd, num nPosition)

9.4. TYP FRAME

9.4.1. DEFINITION

Der Typ frame dient zur Festlegung der Position der Referenz-Koordinatensysteme in der Arbeitszelle.

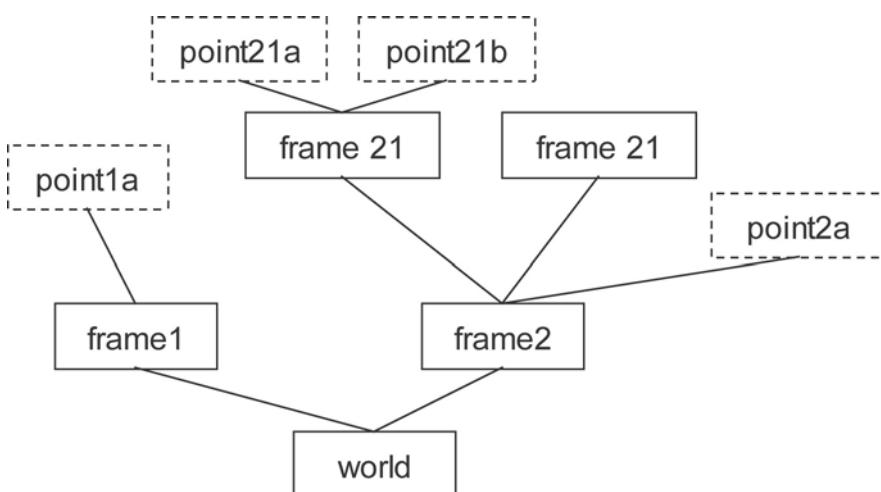
Der Typ frame ist ein strukturierter Typ mit einem einzigen verfügbaren Feld:

trsfrsf trsf Position des Koordinatensystems in seinem Referenz-Koordinatensystem

Das **Referenz-Koordinatensystem** einer Variable des Typs **frame** wird bei seiner Initialisierung festgelegt (von der Benutzerschnittstelle aus, mit dem Operator `=` oder mit der Anweisung `link()`). Das Referenz-Koordinatensystem **world** ist in einer **VAL 3**-Applikation immer definiert: Alle Referenz-Koordinatensysteme sind direkt oder über andere Koordinatensysteme mit **world** verbunden.

Wurden die Koordinaten des Bezugssystems **world** geändert, so wird bei jeder geometrischen Berechnung ein Laufzeitfehler erzeugt.

Zusammenhang zwischen den Referenz-Koordinatensystemen



Standardmäßig haben lokale frame-Variablen und frames in Benutzertyp-Variablen kein vorprogrammiertes Referenz-Koordinatensystem. Um sie zu verwenden, müssen sie von einem anderen Koordinatensystem aus mit dem Operator `=` oder mit einer der Anweisungen `link()` oder `setFrame()` initialisiert werden.

9.4.2. VERWENDUNG

Die Verwendung von Referenz-Koordinatensystemen in einer Roboteranwendung wird ausdrücklich empfohlen:

- **Um die einzelnen Punkte der Applikation intuitiv erkennen zu können**
Die Punkte der Arbeitszelle werden entsprechend der hierarchischen Ordnung der Koordinatensysteme in strukturierter Weise angezeigt.
- **Um die Position einer Gruppe von Punkten rasch zu ändern**
Sobald ein Punkt der Applikation an ein Objekt gebunden ist, sollte für dieses Objekt ein Koordinatensystem definiert und die **VAL 3**-Punkte mit ihm verbunden werden. Wenn das Objekt verschoben wird, braucht nur das Koordinatensystem neu gelernt zu werden, und alle mit ihm verbundenen Punkte werden gleichzeitig korrigiert.
- **Zur Wiederholung einer Bahn an mehreren Stellen der Arbeitszelle**
Hierzu können die Bahnpunkte in Bezug auf ein Arbeitskoordinatensystem festgelegt werden, das dann an jeder Stelle, an der die Bahn wiederholt werden soll, gelernt wird. Durch Zuweisung des Wertes eines gelernten Koordinatensystems an das Arbeitskoordinatensystem wird die gesamte Bahn auf das gelernte Koordinatensystem "verschoben".
- **Um geometrische Verschiebungen leichter berechnen zu können**
Die Anweisung `compose()` erlaubt geometrische Verschiebungen an einen beliebigen Punkt, die in irgendeinem Referenz-Koordinatensystem ausgedrückt werden. Die Anweisung `position()` dient zur Berechnung der Position eines Punktes in einem beliebigen Referenz-Koordinatensystem.

9.4.3. OPERATOREN

Mit zunehmender Priorität:

| | |
|---|--|
| frame <frame& fReference1> = <frame fReference2> | Weist die Position und das Referenz-Koordinatensystem von fReference2 der Variable fReference1 zu. |
| bool <frame fReference1> != <frame fReference2> | Überträgt true , wenn fReference1 und fReference2 nicht das gleiche Referenz-Koordinatensystem oder darin nicht die gleiche Position haben. |
| bool <frame fReference1> == <frame fReference2> | Überträgt true , wenn fReference1 und fReference2 die gleiche Position im gleichen Referenz-Koordinatensystem haben. |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig.

9.4.4. ANWEISUNGEN

**num setFrame(point pOrigin, point pAxisOx, point pPlaneOxy,
frame& fResult)**

Funktion

Diese Anweisung berechnet die Koordinaten von **fResult** aus dem Ursprungspunkt **pOrigin**, eines Punktes **pAxisOx** auf der **(Ox)**-Achse und eines Punktes **pPlaneOxy** in der Ebene **(Oxy)**.

Der Punkt **pAxisOx** muss auf der positiven **x**-Achse liegen. Der Punkt **pPlaneOxy** muss auf der positiven **y**-Achse liegen.

Die Funktion liefert folgende Antwort:

| | |
|-----------|--|
| 0 | Kein Fehler. |
| -1 | Der Punkt pAxisOx liegt zu dicht an pOrigin . |
| -2 | Der Punkt pPlaneOxy liegt zu dicht an der Achse (Ox) . |

Wenn einer der Punkte kein Referenz-Koordinatensystem besitzt, wird ein Laufzeitfehler erzeugt.

trsf position(frame fFrame, frame fReference)

Funktion

Die Anweisung liefert die Koordinaten des Frame **fFrame** bezüglich des Frames **fReference**.

Wenn **fFrame** oder **fReference** kein Referenz-Koordinatensystem besitzen, wird ein Laufzeitfehler erzeugt.

Siehe auch

trsf position(point pPosition, frame fReference)
trsf position(tool tTool, tool tReference)

void link(frame fFrame, frame fReference)

Funktion

Diese Anweisung ändert das Referenz-Koordinatensystem von **fFrame** und verbindet es mit **fReference**. Die Position des Koordinatensystems in seinem Referenz-Koordinatensystem bleibt unverändert.

Siehe auch

Operator frame <frame& fFrame1> = <frame fFrame2>

9.5. TYP TOOL

9.5.1. DEFINITION

Der Typ **tool** dient zur Definition der Geometrie und der Aktion eines Werkzeugs.

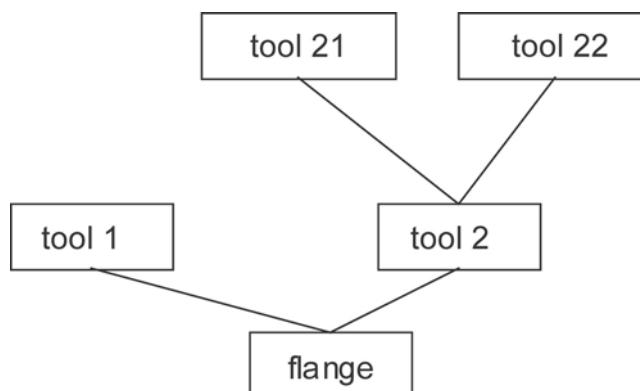
Der Typ **tool** ist ein strukturierter Typ mit folgenden Feldern in der nachstehenden Reihenfolge:

| | |
|--------------------|--|
| trsfp trsf | Position des Werkzeugnullpunktes (TCP) bezüglich seines Basiswerkzeugs |
| dio gripper | Digitaler Ausgang zur Steuerung der Aktion des Werkzeugs |
| num otim | Öffnungszeit des Werkzeugs (in Sekunden) |
| num ctime | Schließzeit des Werkzeugs (in Sekunden) |

Das **Referenz-Tool** einer Variable des Typs **tool** wird bei ihrer Initialisierung festgelegt (von der Benutzerschnittstelle aus, mit dem Operator `=` oder mit der Anweisung [link\(\)](#)). Das Tool **flange** ist in einer **VAL 3**-Applikation immer definiert: Jedes Tool ist direkt oder über andere Tools mit dem Tool **flange** verbunden.

Wurden die Koordinaten des Werkzeugs **flange** geändert, so wird bei jeder geometrischen Berechnung ein Laufzeitfehler erzeugt.

Abhängigkeit der Tools



Der Ausgang eines Tools ist vorprogrammiert auf Ausgang **valve1** des Systems, die Öffnungs- und Schließzeit sind **0** und das Basistool ist **flange**. Lokale Tool-Variablen und Tools in Benutzertyp-Variablen haben kein vorprogrammiertes Referenz-Tool. Um sie zu verwenden, müssen sie von einem anderen Tool aus mit dem Operator `'='` oder der Anweisung [link\(\)](#) initialisiert werden.

9.5.2. VERWENDUNG

Die Verwendung von Werkzeugen in einer Roboteranwendung wird ausdrücklich empfohlen:

- **Um die Bewegungsgeschwindigkeit zu steuern**
Bei manuellen oder programmierten Bewegungen steuert das System die kartesische Geschwindigkeit am Werkzeugende.
- **Um mit verschiedenen Werkzeugen die gleichen Punkte anzufahren**
Hierzu genügt es, das **VAL 3**-Werkzeug zu wählen, das dem am Arm montierten realen Werkzeug entspricht.
- **Um den Werkzeugverschleiß oder einen Werkzeugwechsel zu verwalten**
Zur Kompensierung des Werkzeugverschleiß müssen nur die Werkzeugkoordinaten aktualisiert werden.

9.5.3. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|--|
| tool <tool& tTool1> = <tool tTool2> | Weist der Variablen tTool1 die Position und das Basiswerkzeug von tTool2 zu. |
| bool <tool tTool1> != <tool tTool2> | Überträgt true , wenn tTool1 und tTool2 nicht das gleiche Basiswerkzeug, die gleiche Position in ihrem Basiswerkzeug, den gleichen digitalen Ausgang oder die gleichen Öffnungs- und Schließzeiten haben. |
| bool <tool tTool1> == <tool tTool2> | Überträgt true , wenn tTool1 und tTool2 die gleiche Position im gleichen Basiswerkzeug, den gleichen digitalen Ausgang mit den gleichen Öffnungs- und Schließzeiten haben. |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig.

9.5.4. ANWEISUNGEN

void open(tool tTool)

Funktion

Diese Anweisung betätigt das Werkzeug (öffnen) indem der digitale Ausgang des Werkzeugs auf **true** gesetzt wird.

Bevor das Tool bewegt werden kann, wartet **open()** bis der Roboter den Punkt erreicht hat, was mittels einer gleichwertigen Aktion wie **waitEndMove()** geschieht. Nach der Aktivierung wartet das System **otime** Sekunden, bevor die nächste Anweisung ausgeführt wird.

Diese Anweisung stellt nicht die Stabilisierung des Roboters vor der Aktivierung des Werkzeugs an seiner Endposition sicher. Wenn die vollständige Stabilisierung nach der Bewegung abgewartet werden soll, so muss die Anweisung **isSettled()** verwendet werden.

Wenn **dio** von **tTool** nicht definiert oder kein Ausgang ist, wird ein Laufzeitfehler erzeugt, ebenso, wenn ein zuvor aufgezeichneter Bewegungsbefehl nicht ausgeführt werden kann (Endpunkt außer Reichweite).

Beispiel

```
// the open() instruction is equivalent to:  
waitEndMove()  
tTool.gripper=true  
delay(tTool.otime)
```

Siehe auch

void close(tool tTool)
void waitEndMove()

void close(tool tTool)

Funktion

Diese Anweisung betätigt das Werkzeug (schließen) indem der digitale Ausgang des Werkzeugs auf **false** gesetzt wird.

Bevor das Tool bewegt werden kann, wartet **close()** bis der Roboter den Punkt erreicht hat, was mittels einer gleichwertigen Aktion wie **waitEndMove()** geschieht. Nach der Aktivierung wartet das System ctime Sekunden, bevor die nächste Anweisung ausgeführt wird.

Diese Anweisung stellt nicht die Stabilisierung des Roboters vor der Aktivierung des Werkzeugs an seiner Endposition sicher. Wenn die vollständige Stabilisierung nach der Bewegung abgewartet werden soll, so muss die Anweisung **isSettled()** verwendet werden.

Wenn **dio** von **tTool** nicht definiert oder kein Ausgang ist, wird ein Laufzeitfehler erzeugt, ebenso, wenn ein zuvor aufgezeichneter Bewegungsbefehl nicht ausgeführt werden kann (Endpunkt außer Reichweite).

Beispiel

```
// the close instruction is equivalent to:  
waitEndMove()  
tTool.gripper = false  
delay(tTool.ctime)
```

Siehe auch

Type tool

void open(tool tTool)
void waitEndMove()

trsf position(tool tTool, tool tReference)

Funktion

Diese Anweisung liefert die Koordinaten des Tools **tTool** relativ zum Tool **tReference**.

Wenn **tTool** oder **tReference** kein Referenz-Tool besitzen, wird ein Laufzeitfehler erzeugt.

Siehe auch

trsf position(point pPosition, frame fReference)
trsf position(frame fFrame, frame fReference)

void link(tool tTool, tool tReference)

Funktion

Diese Anweisung ändert das Referenz-Tool von **tTool** und verbindet es mit **tReference**. Die Position des Tools in seinem Referenz-Tool bleibt unverändert.

Siehe auch

Operator tool <tool& tTool1> = <tool tTool2>

9.6. TYP POINT

9.6.1. DEFINITION

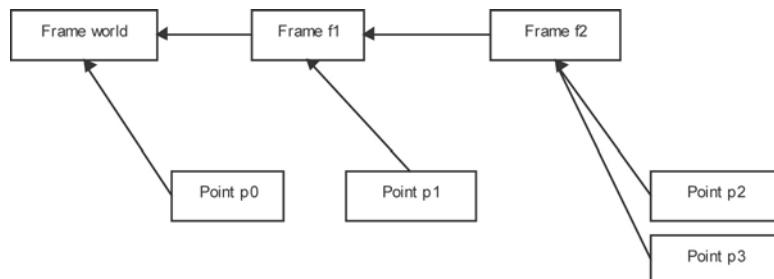
Der Typ **point** dient zur Definition der Position und Orientierung des Roboterwerkzeugs in der Arbeitszelle.

Der Typ **point** ist ein strukturierter Typ mit folgenden Feldern in der nachstehenden Reihenfolge:

| | |
|----------------------|--|
| trsf trTrsf | Position des Punktes in seinem Referenz-Koordinatensystem |
| config config | Konfiguration des Arms, um die Position anfahren zu können |

Das Referenz-Koordinatensystem eines **point** ist eine Variable des Typs **frame**, die bei ihrer Initialisierung definiert wird (von der Benutzerschnittstelle aus, mit dem Operator **=** oder den Anweisungen **link()**, **here()**, **appro()** und **compose()**).

Punktdefinition



Wenn man eine Variable des Typs **point** ohne definiertes Referenz-Frame verwendet, wird ein Laufzeitfehler erzeugt.

ACHTUNG:

Standardmäßig haben lokale **point**-Variablen und **points** in Benutzertyp-Variablen kein vorprogrammiertes Referenz-Frame. Um sie zu verwenden, müssen sie von einem anderen **point** aus mit dem Operator **'='** oder mit einer der Anweisungen **link()**, **here()**, **appro()** oder **compose()** initialisiert werden.

9.6.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|---|--|
| point <point& pPoint1> = <point pPoint2> | Weist die Position, die Konfiguration und das Referenz-Koordinatensystem von pPoint1 der Variablen pPoint2 zu. |
| bool <point pPoint1> != <point pPoint2> | Überträgt true , wenn pPoint1 und pPoint2 nicht das gleiche Referenz-Koordinatensystem oder darin nicht die gleiche Position haben. |
| bool <point pPoint1> == <point pPoint2> | Überträgt true , wenn pPoint1 und pPoint2 die gleiche Position im gleichen Referenz-Koordinatensystem haben. |

Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete **VAL 3**-Ausdrücke nicht zulässig.

9.6.3. ANWEISUNGEN

num distance(point pPosition1, point pPosition2)

Funktion

Diese Anweisung liefert die Entfernung zwischen **pPosition1** und **pPosition2**.

Ein Laufzeitfehler wird erzeugt, wenn das Referenz-Koordinatensystem von **pPosition1** oder **pPosition2** nicht festgelegt ist.

Beispiel

Dieses Programm wartet, bis der Arm sich näher als 10 mm an der Position pTarget befindet

```
wait(distance(here(tTool,world),pTarget)< 10)
```

Siehe auch

point appro(point pPosition, trsf trTransformation)
point compose(point pPosition, frame fReference, trsf trTransformation)
trsf position(point pPosition, frame fReference)
num distance(trsf trPosition1, trsf trPosition2)

**point compose(point pPosition, frame fReference,
trsf trTransformation)**

Funktion

Diese Anweisung liefert **pPosition**, auf welche die in Bezug auf **fReference** definierte Koordinatentransformation **trTransformation** angewandt wurde.

ACHTUNG:

Die Orientierung von **tTransformation** ändert in der Regel nicht nur die Ausrichtung von **pPosition**, sondern auch seine kartesischen Koordinaten (außer wenn sich **pPosition** am Ursprung von **fReference** befindet).

Wenn **tTransformation** nur die Ausrichtung von **pPosition** ändern soll, muss das Ergebnis mit den kartesischen Koordinaten von **pPosition** aktualisiert werden (siehe Beispiel).

Das Referenz-Koordinatensystem und die Konfiguration des übertragenen Punktes sind die von **pPosition**.

Wenn für **pPosition** kein Referenz-Koordinatensystem definiert ist, wird ein Laufzeitfehler erzeugt.

Beispiel

```
// modification of the orientation without modification of Position
pResult = compose(pPosition,fReference,trTransformation)
pResult.trsf.x = pPosition.trsf.x
pResult.trsf.y = pPosition.trsf.y
pResult.trsf.z = pPosition.trsf.z
// modification of Position without modification of the orientation
trTransformation.rx = trTransformation.ry = trTransformation.rz = 0
pResult = compose(pResult,fReference,trTransformation)
```

Siehe auch

Operator trsf <trsf pPosition1> * <trsf pPosition2>
point appro(point pPosition, trsf trTransformation)

point appro(point pPosition, trsf trTransformation)

Funktion

Die Anweisung liefert einen durch eine Koordinatentransformation geänderten point. Die Transformation wird relativ zur Orientierung des Eingabepunkts definiert.

Das Referenz-Koordinatensystem und die Konfiguration des ermittelten Punktes entsprechen denen des Eingabepunkts.

Wenn für pPosition kein Referenz-Koordinatensystem definiert ist, wird ein Laufzeitfehler erzeugt.

Beispiel

```
// Methode:move to 100 mm above the point (Z axis)
movej(appro(pDestination,{0,0,-100,0,0,0}), flange, mNomDesc)
// Go to point
movel(pDestination, flange, mNomDesc)
```

Siehe auch

Operator trsf <trsf trPosition1> * <trsf trPosition2>
point compose(point pPosition, frame fReference, trsf trTransformation)

point here(tool tTool, frame fReference)

Funktion

Diese Anweisung liefert die aktuelle Position des Werkzeugs **tTool** in **fReference** (Sollposition). Das Referenz-Koordinatensystem des übertragenen Punktes ist **fReference**. Die Konfiguration des übertragenen Punktes ist die aktuelle Konfiguration des Arms.

Siehe auch

```
joint herej()
config config(joint jPosition)
point jointToPoint(tool tTool, frame fReference, joint jPosition)
```

point jointToPoint(tool tTool, frame fReference, joint jPosition)

Funktion

Diese Anweisung liefert die Position des Tools **tTool** in **fReference**, wenn der Arm in der Joint-Position **jPosition** ist.

Das Referenz-Koordinatensystem des übertragenen Punktes ist **fReference**. Die Konfiguration des zurückgegebenen Punktes ist die des Arms in der Joint-Position **jPosition**.

Siehe auch

```
point here(tool tTool, frame fReference)
bool pointToJoint(tool tTool, joint jInitial, point pPosition, joint& jResult)
```

**bool pointToJoint(tool tTool, joint jInitial, point pPosition,
joint& jResult)**

Funktion

Diese Anweisung berechnet die dem angegebenen Punkt **pPosition** entsprechende Gelenkposition **jResult**. Liefert **true**, wenn **jResult** aktualisiert wird oder **false** wenn keine Lösung gefunden wurde.

Die gesuchte Joint-Position ist mit der Konfiguration von **pPosition** kompatibel. Die Felder mit dem Wert **free** schreiben keine Konfiguration vor. Die Felder mit dem Wert **same** schreiben die gleiche Konfiguration wie **jInitial** vor.

Für Achsen, die mehr als eine Umdrehung ausführen können, gibt es mehrere Joint-Lösungen, die genau die gleiche Konfiguration haben: Es wird die **jInitial** am nächsten liegende Lösung genommen.

Wenn **pPosition** außer Reichweite (Arm zu kurz) oder außerhalb der Softwaregrenzen liegt, kann es sein, dass es keine Lösung gibt. Wenn **pPosition** eine Konfiguration spezifiziert, kann sie für diese Konfiguration außerhalb, aber für eine andere Konfiguration innerhalb der Softwaregrenzen liegen.

Wenn für **pPosition** kein Referenz-Koordinatensystem definiert ist, wird ein Laufzeitfehler erzeugt.

Siehe auch

joint herej()
point jointToPoint(tool tTool, frame fReference, joint jPosition)

trsf position(point pPosition, frame fReference)

Funktion

Diese Anweisung liefert die Koordinaten von **pPosition** im Koordinatensystem **fReference**.

Wenn **pPosition** kein Referenz-Koordinatensystem besitzt, wird ein Laufzeitfehler erzeugt.

Beispiel

Die Entfernung zwischen 2 Punkten, ist die Entfernung zwischen ihren Positionen in world:

```
distance(position(pPoint1, world), position(pPoint2, world)) is distance(pPoint1, pPoint2)
```

Siehe auch

num distance(point pPosition1, point pPosition2)
trsf position(tool tTool, tool tReference)
trsf position(frame fFrame, frame fReference)

void link(point pPoint, frame fReference)

Funktion

Diese Anweisung ändert das Referenz-Koordinatensystem von **pPoint** und verbindet es mit **fReference**. Die Position des Punkts im Referenz-Koordinatensystem bleibt unverändert.

Siehe auch

Operator **point <point& pPoint1> = <point pPoint2>**

9.7. TYP CONFIG

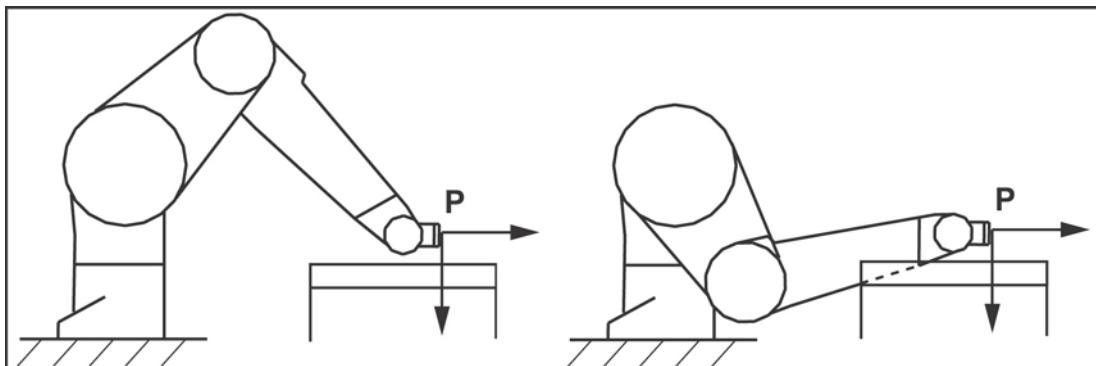
Die Konfiguration eines kartesischen Punktes gehört zu den komplexeren Funktionen; dieses Kapitel kann beim ersten Lesen überschlagen werden.

9.7.1. EINLEITUNG

Im Allgemeinen hat der Roboter mehrere Möglichkeiten, eine in kartesischen Koordinaten angegebene Position anzufahren.

Diese verschiedenen Möglichkeiten werden "Konfigurationen" genannt. Die nachfolgende Zeichnung zeigt zwei verschiedene Konfigurationen:

Zwei verschiedene Konfigurationen, um denselben Punkt zu erreichen: P



In bestimmten Fällen ist es wichtig, anzugeben, welche der möglichen Konfigurationen zulässig sind und welche verboten werden sollen. Um dieses Problem zu lösen, können mit dem Typ **point** dank des nachstehend definierten Feldes des Typs **config** die zulässigen Roboterkonfigurationen spezifiziert werden.

9.7.2. DEFINITION

Typ **config** ermöglicht die Definition der erlaubten Konfigurationen für eine bestimmte kartesische Position.

Er hängt vom verwendeten Armtyp ab.

Für einen **Stäubli RX/TX-Arm** ist der Typ **config** ein strukturierter Typ mit folgenden Feldern in der nachstehenden Reihenfolge:

| | |
|-----------------|-------------------------------|
| shoulder | Konfiguration der Schulter |
| elbow | Konfiguration des Ellenbogens |
| wrist | Konfiguration des Handgelenks |

Für einen **Stäubli RS/TS-Arm** ist der Typ **config** auf das Feld **Shoulder** begrenzt:

| | |
|-----------------|----------------------------|
| shoulder | Konfiguration der Schulter |
|-----------------|----------------------------|

Die Felder **shoulder**, **elbow** und **wrist** können folgende Werte annehmen:

| | | |
|-----------------|------------------|---|
| shoulder | righty | Konfiguration der Schulter righty vorgeschrieben |
| | lefty | Konfiguration der Schulter lefty vorgeschrieben |
| | ssame | Andere Schulterkonfiguration verboten |
| | sfree | Beliebige Konfiguration der Schulter |
| elbow | epositive | Konfiguration des Ellenbogens epositive vorgeschrieben |
| | enegative | Konfiguration des Ellenbogens enegative vorgeschrieben |
| | esame | Andere Ellenbogenkonfiguration verboten |
| | efree | Beliebige Konfiguration des Ellenbogens |
| wrist | wpositive | Konfiguration des Handgelenks wpositive vorgeschrieben |
| | wnegative | Konfiguration des Handgelenks wnegative vorgeschrieben |
| | wsame | Andere Handgelenkkonfiguration verboten |
| | wfree | Beliebige Konfiguration des Handgelenks |

9.7.3. OPERATOREN

Mit zunehmender Priorität:

| | |
|--|--|
| config <config& configuration1> = <config configuration2> | Weist die Felder shoulder , elbow und wrist von configuration2 der Variablen configuration1 zu. |
| bool <config configuration1> != <config configuration2> | Überträgt true , wenn die Felder shoulder , elbow oder wrist in configuration1 und configuration2 nicht den gleichen Wert haben. |
| bool <config configuration1> == <config configuration2> | Überträgt true , wenn die Felder shoulder , elbow oder wrist in configuration1 und configuration2 den gleichen Wert haben. |

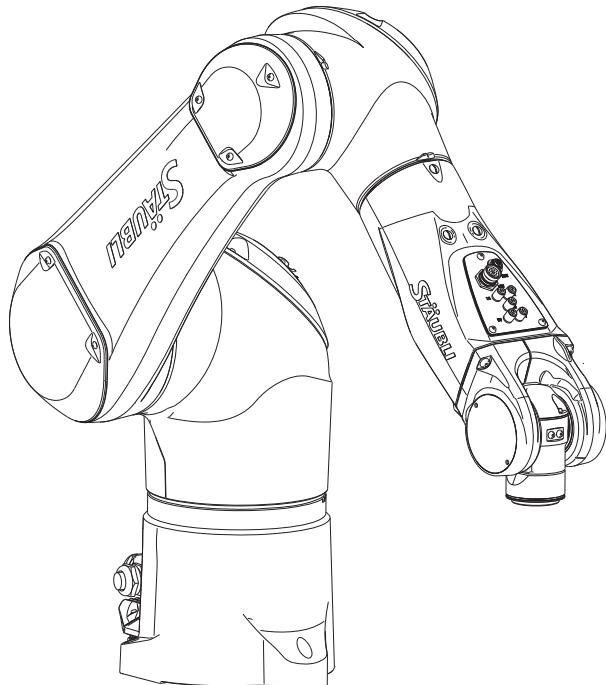
Um Verwechslungen zwischen den Operatoren **=** und **==** zu vermeiden, ist der Operator **=** für als Anweisungsparameter verwendete VAL 3-Ausdrücke nicht zulässig.

9.7.4. KONFIGURATION (ARM RX/TX)

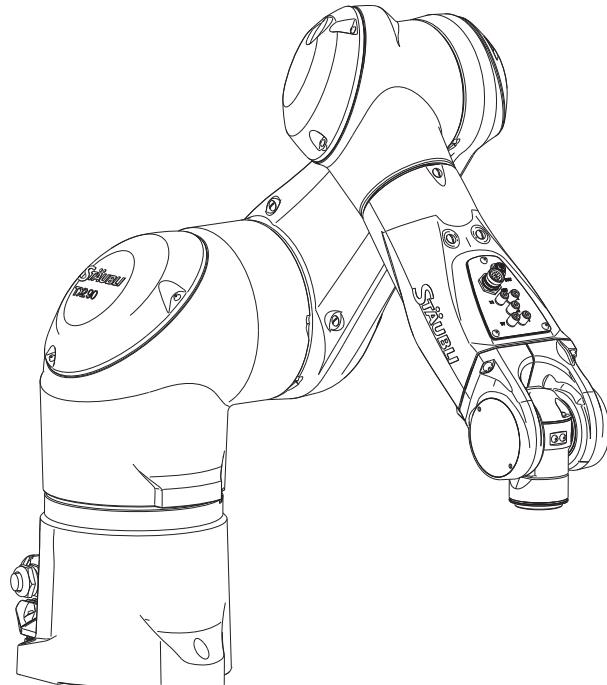
9.7.4.1. KONFIGURATION DER SCHULTER

Um einen kartesischen Punkt zu erreichen kann sich der Roboterarm rechts oder links von diesem Punkt befinden: Diese beiden Konfigurationen werden **righty** und **lefty** genannt.

Konfiguration: righty



Konfiguration: lefty

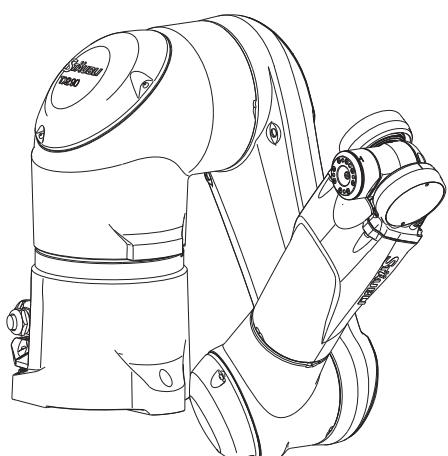


Die Konfiguration **righty** ist definiert durch $((d1 * \sin(j2) + d2 * \sin(j2+j3) + \delta) < 0$, die Konfiguration **lefty** durch $((d1 * \sin(j2) + d2 * \sin(j2+j3) + \delta) \geq 0$, mit $d1$ = Armlänge des Roboters, $d2$ = Unterarmlänge und δ = Abstand der Achsen 1 und 2 in Richtung x.

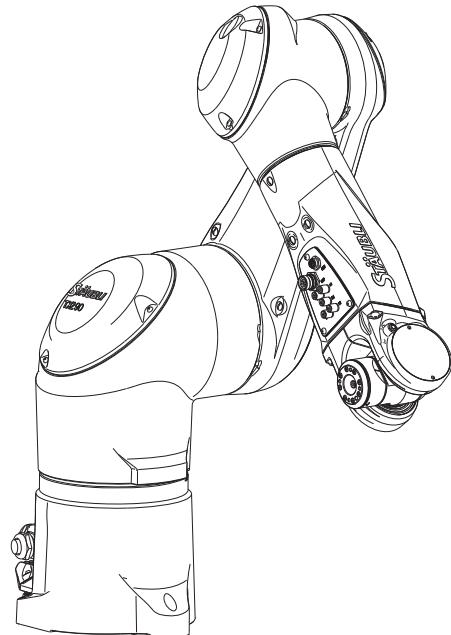
9.7.4.2. KONFIGURATION DES ELLENBOGENS

Zusätzlich zur Konfiguration der Schulter gibt es zwei Möglichkeiten für das Ellenbogengelenk des Roboters: Die Ellenbogenkonfigurationen werden **epositive** und **enegative** genannt.

Konfiguration: enegative



Konfiguration: epositive

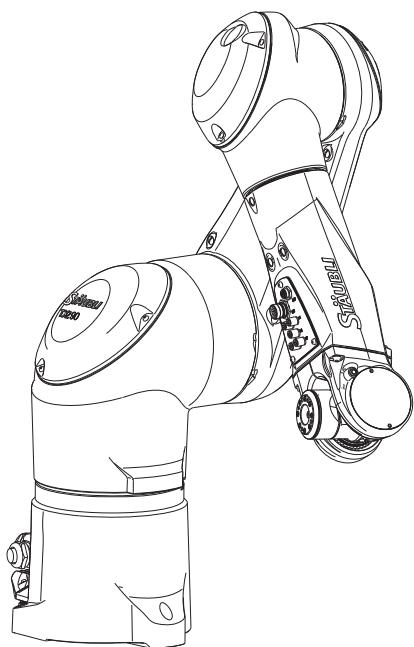
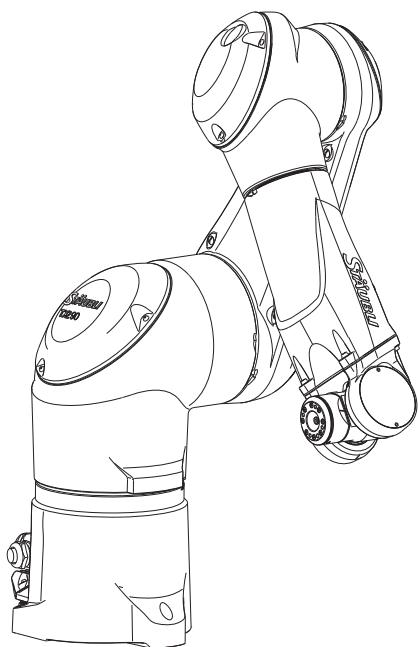


Die Konfiguration **epositive** ist durch $j3 \geq 0$ definiert.

Die Konfiguration **enegative** ist durch $j3 < 0$ definiert.

9.7.4.3. KONFIGURATION DES HANDGELENKS

Außer der Konfiguration der Schulter und des Ellenbogens gibt es zwei Möglichkeiten für das Handgelenk des Roboters. Die beiden Konfigurationen des Handgelenks werden **wpositive** und **wnegative** genannt.

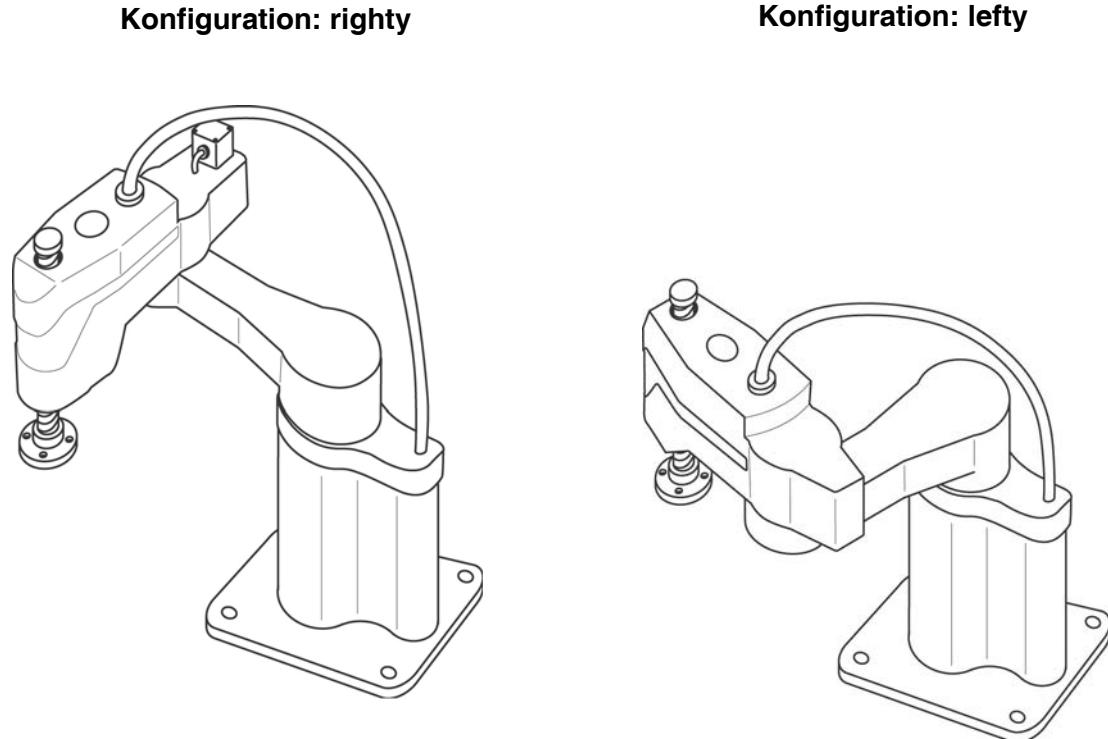
Konfiguration: wnegative**Konfiguration: wpositive**

Die Konfiguration **wpositive** ist durch $j5 \geq 0$ definiert.

Die Konfiguration **wnegative** ist durch $j5 < 0$ definiert.

9.7.5. KONFIGURATION SCARA (TS/TP ARM)

Um einen kartesischen Punkt zu erreichen kann sich der Roboterarm rechts oder links von diesem Punkt befinden: Diese beiden Konfigurationen werden **righty** und **lefty** genannt.



Die Konfiguration **righty** ist definiert durch $\sin(j2) > 0$, die Konfiguration **lefty** durch $\sin(j2) \leq 0$.

9.7.6. ANWEISUNGEN

config config(joint jPosition)

Funktion

Diese Anweisung liefert die Roboterkonfiguration für die Joint-Position **jPosition**.

Siehe auch

point here(tool tTool, frame fReference)
joint herej()

KAPITEL 10

BEWEGUNGSSTEUERUNG

10.1. BEWEGUNGSSTEUERUNG

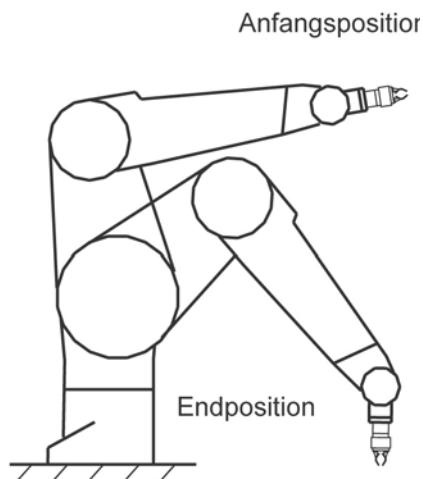
Um eine Roboterbahn zu definieren, genügt es nicht, nur eine Reihe von Punkten anzugeben. Es müssen außerdem die Art der zwischen diesen Punkten verwendeten Bahnen (Kurven- oder Geradenabschnitte) sowie die Art der Übergänge zwischen diesen Abschnitten und schließlich die Geschwindigkeitsparameter für diese Bewegung festgelegt werden. Dieses Kapitel gibt eine Einführung in die verschiedenen Bewegungstypen (Anweisungen **movej**, **movel** und **movec**) und beschreibt, wie die Parameter des Motion descriptors (Typ **mdesc**) zu verwenden sind.

10.1.1. BEWEGUNGSTYPEN: PUNKT-ZU-PUNKT, GERADLINIG, KREISFÖRMIG

Roboterbewegungen werden vor allem mit den Anweisungen **movej**, **movel** und **movec** programmiert. Mit **movej** werden Punkt-zu-Punkt-Bewegungen, mit **movel** geradlinige Bewegungen und mit **movec** kreisförmige Bewegungen ausgeführt.

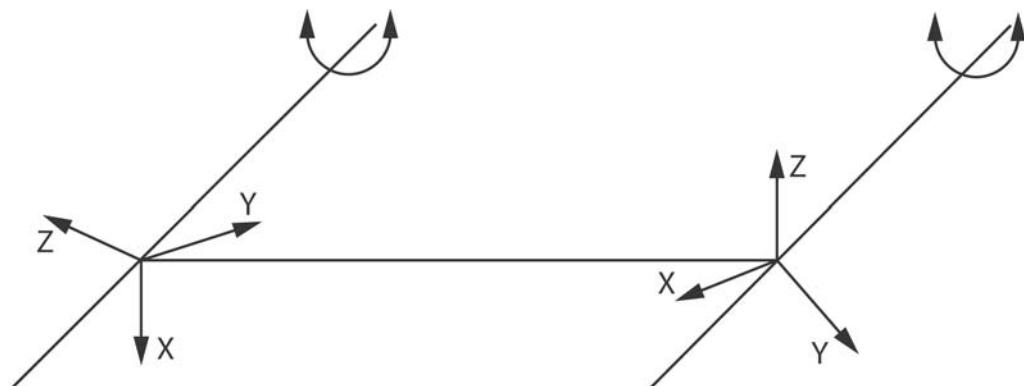
Bei einer Punkt-zu-Punkt-Bewegung ist nur der Endpunkt (kartesische oder Joint-Position) von Bedeutung. Zwischen dem Anfangs- und dem Endpunkt folgt das Werkzeugzentrum einer vom System definierten Kurve, um die Bewegungsgeschwindigkeit zu optimieren.

Anfangs- und Endposition



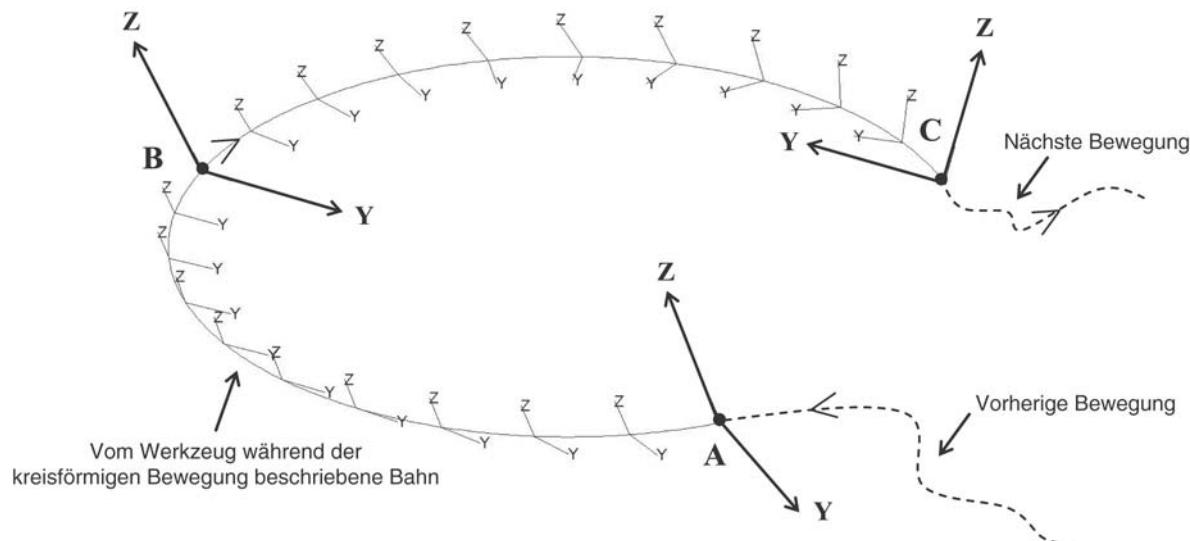
Dagegen bewegt sich der Werkzeugmittelpunkt bei einer geradlinigen Bewegung entlang der vorgegebenen Geraden. Die Orientierung wird zwischen der Ausgangs- und Endorientierung des Werkzeugs linear interpoliert.

Geradlinige Bewegung



Bei einer kreisförmigen Bewegung bewegt sich die Werkzeugmitte entlang eines durch **3** Punkte definierten Kreisbogens, und die Orientierung des Werkzeugs wird zwischen Ausgangs-, Mittel- und Endorientierung interpoliert.

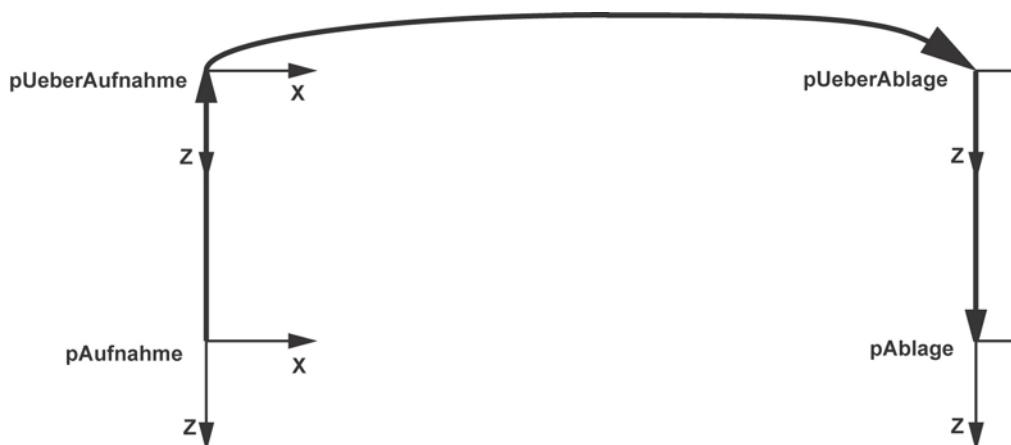
Kreisförmige Bewegung



Beispiel:

Eine typische Handlingaufgabe besteht darin, Teile an einem Ort aufzunehmen und an einem anderen Ort abzulegen. Angenommen, die Teile sollen am Punkt **pPick** aufgenommen und am Punkt **pPlace** abgelegt werden. Um vom Punkt **pPick** zum Punkt **pPlace** zu gelangen, muss der Roboter über einen Zwischenpunkt **pDepart** und einen Annäherungspunkt **pAppro** fahren.

Zyklus: U



Angenommen, der Roboter steht zu Beginn am Punkt **pPick**. Dann kann das Programm für diese Bewegung wie folgt aussehen:

```
moveL(pDepart, tTool, mDesc)
moveJ(pAppro, tTool, mDesc)
moveL(pPlace, tTool, mDesc)
```

Zum Freifahren und zur Annäherung werden gerade Bahnabschnitte verwendet. Die Hauptbewegung ist dagegen eine Punkt-zu-Punkt-Bewegung, da dieser Teil der Bahn keine genaue geometrische Steuerung erfordert, sondern vielmehr möglichst rasch zurückgelegt werden soll.

Hinweis:

Bei beiden Bewegungstypen hängt die Geometrie der Bahn nicht von der Geschwindigkeit ab, mit der die Bewegungen ausgeführt werden. Der Roboter fährt immer den gleichen Punkt an. Dieser Umstand ist bei der Entwicklung von Applikationen besonders wichtig. Man kann mit langsam Bewegungen beginnen und die Geschwindigkeit nach und nach erhöhen, ohne dabei die Bahn des Roboters zu ändern.

10.1.2. VERKETTUNG VON BEWEGUNGEN: BLENDING

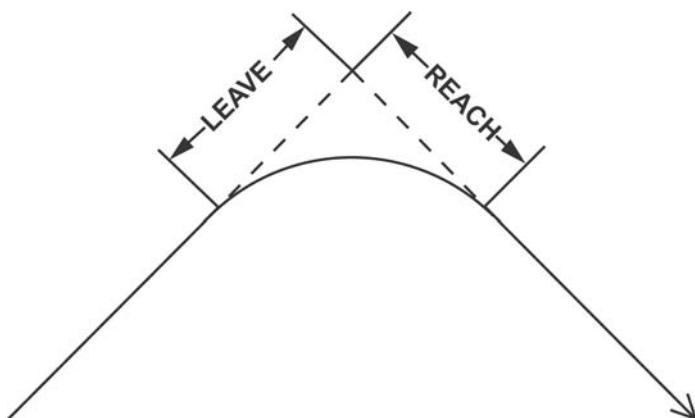
10.1.2.1. BLENDING

Sehen wir uns das Beispiel des **U**-Zyklus im letzten Abschnitt an. Ohne besondere Maßnahmen zur Verkettung der Bewegungen würde der Roboter an den Punkten **pDepart** und **pAppro** anhalten, denn die Bahn hat an diesen Stellen einen Knick. Das würde unnötig Zeit kosten, außerdem ist das genaue Anfahren dieser Punkte gar nicht erforderlich.

Durch Glättung der Bahn in der Umgebung der Punkte **pDepart** und **pAppro** lässt sich die Dauer der gesamten Bewegung erheblich verkürzen. Hierzu dient das Feld **blend** im Bewegungsdeskriptor. Wenn der Wert in diesem Feld auf **off** steht, hält der Roboter an jedem Punkt an. Wenn dieser Parameter jedoch auf **joint** oder **Cartesian** gesetzt wird, wird die Bahn in der Umgebung dieser Punkte geglättet und der Roboter hält beim Überfahren der Punkte nicht mehr an.

Wenn das Feld **blend** den Wert **joint** oder **Cartesian** annimmt, müssen zwei weitere Parameter angegeben werden: **leave** und **reach**. Mit diesen Parametern wird festgelegt, in welcher Entfernung vom Zielpunkt die theoretische Bahn verlassen (Beginn der Glättung) und in welcher Entfernung vom Zielpunkt sie wieder eingenommen wird (Ende der Glättung).

Definition der Entfernungen: 'leave' / 'reach'

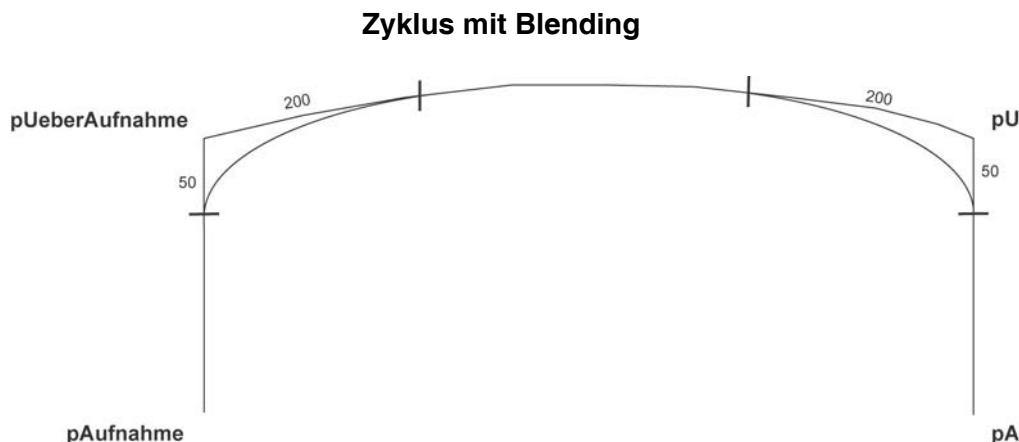


Beispiel:

Betrachten wir das Programm im Kapitel "Bewegungstypen:Punkt-zu-Punkt und geradlinig". Das vorhergehende Bewegungsprogramm kann geändert werden:

```
mDesc.blend = joint
mDesc.leave = 50
mDesc.reach = 200
moveL(pDepart, tTool, mDesc)
mDesc.leave = 200
mDesc.reach = 50
moveJ(pAppro, tTool, mDesc)
mDesc.blend = off
moveL(pPlace, tTool, mDesc)
```

Damit erhält man folgende Bahn:



Der Roboter hält an den Punkten **pDepart** und **pAppro** nicht mehr an. Die Bewegung wird daher schneller. Je größer die Entferungen **leave** und **reach** gewählt werden, desto rascher wird die Bewegung.

10.1.2.2. BLENDING AUFHEBEN

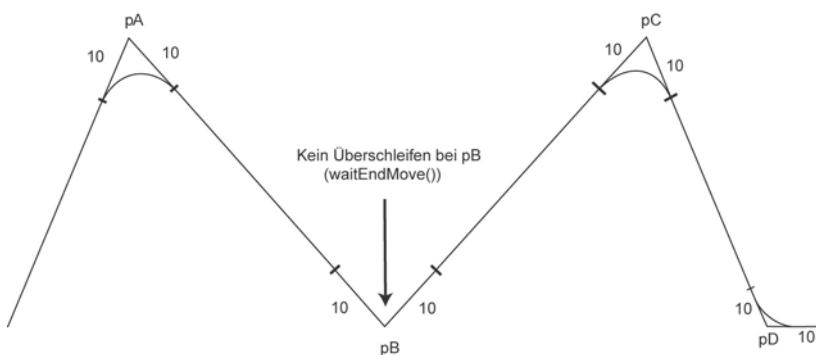
Die Anweisung **waitEndMove()** ermöglicht unter anderem, das Blending aufzuheben. In diesem Fall führt der Roboter die zuletzt programmierte Bewegung bis zum Zielpunkt aus, als ob das Feld **blend** im Bewegungsdeskriptor auf **off** stehen würde.

Betrachten wir zum Beispiel das folgende Programm:

```
mDesc.blend = joint
mDesc.leave = 10
mDesc.reach = 10
movej(pA, tTool, mDesc)
movej(pB, tTool, mDesc)
waitEndMove()
movej(pC, tTool, mDesc)
movej(pD, tTool, mDesc)
etc...
```

ergibt sich folgende Bahn des Roboters:

Zyklus ohne Blending an einem Punkt



10.1.2.3. JOINT BLENDING, KARTESISCHES BLENDING

Vereinfacht beschrieben, ist ein Joint-Blending eine Punkt-zu-Punkt-Bewegung zwischen Ausgangs- und Zielpunkt und das kartesische Blending ist wie eine kreisförmige Bewegung zwischen diesen Punkten.

- Joint-Blending ist normalerweise schneller als kartesisches Blending. Aber Joint-Blending kann ungewöhnliche Bahnen zur Folge haben, wenn es zu komplexen Richtungsänderungen kommt (typischerweise eine Kreisbewegung in einer Ebene gefolgt von einer Kreisbewegung in einer orthogonalen Ebene) oder bei reinen

Rotationsbewegungen.

- Geschwindigkeits- und Beschleunigungskontrolle sind mit kartesischem Blending genauer. Eine kartesisches Blending zwischen zwei Bewegungen in derselben Ebene erfolgt garantiert in dieser Ebene.

Die am besten optimierte Form für das Blending ist von der Applikation abhängig, aber der **VAL 3**-Interpreter muss die Form automatisch wählen. Das Ergebnis ist normalerweise nicht überraschend, aber manchmal eben doch ...

Die Auswahl kann zu einer unerwarteten komplexen Form führen, wenn die effektiven Leave- und Reachwerte sehr unterschiedlich sind. Die berechnete Form reduziert die Krümmung des Pfads zur Geschwindigkeitsoptimierung, aber das Ergebnis ist für manche Prozessapplikationen nicht wünschenswert. Wenn die Leave- und Reachwerte gleich sind, führt die kartesische Glättung stets zu einer einfachen Form.

Das kartesische Blending gilt für Position und Orientierung. Ein komplexer Richtungswechsel kann sich auf die Form der Glättung auswirken und unerwartete Ergebnisse liefern. Es gibt auch einige Beschränkungen mit Richtungswechsel: wie in einem Kreis, große Richtungswechsel führen zu Bewegungsfehlern, wenn mehrere Lösungen möglich sind, aber das System keine Kriterien besitzt um eines auszuwählen. In diesem Fall sind zusätzliche Zwischenpunkt(e) notwendig, um dem System zu helfen die korrekte Interpolationsrichtung zu finden.

10.1.3. WIEDERAUFAHME EINER BEWEGUNG

Wird die Energieversorgung des Arms unterbrochen bevor der Roboter seine Bewegungen beendet hat, zum Beispiel nach einer Notabschaltung, so muss nach der Wiederherstellung der Energieversorgung eine Bewegungs-Wiederaufnahme durchgeführt werden. Wurde der Arm während der Abschaltung per Hand verschoben, so kann er von seiner Bahn weit entfernt sein. Es muss also sichergestellt werden, dass die Bewegung ohne Kollisionsgefahr wiederaufgenommen werden kann. Die **VAL 3**-Bahnkontrolle ermöglicht die Verwaltung der Bewegungs-Wiederaufnahme mittels einer "Anschlussbewegung".

Bei der Wiederaufnahme der Bewegung stellt das System sicher, dass sich der Roboter auf der programmierten Bahn befindet: Bei Abweichungen (auch sehr kleinen Werts) zeichnet er automatisch einen Punkt-zu-Punkt-Bewegungsbefehl auf, der zur exakten Position, aus welcher der Roboter seine Bahn verlassen hat, zurückführt: Dies ist die sogenannte "Anschlussbewegung". Sie erfolgt mit reduzierter Geschwindigkeit. Sie muss vom Bediener bestätigt werden, außer bei Automatikbetrieb, wo sie ohne menschliche Eingriffe erfolgen kann. Das Verhalten bei Automatikbetrieb kann mittels der Anweisung **autoConnectMove()** festgelegt werden.

Die Anweisung **resetMotion()** ermöglicht das Abbrechen der aktuellen Bewegung und das eventuelle Programmieren einer Anschlussbewegung, mit deren Hilfe eine Position bei reduzierter Geschwindigkeit unter der Kontrolle des Bedieners angefahren werden kann.

10.1.4. BESONDERHEITEN DER KARTESISCHEN BEWEGUNGEN (GERADLINIG, KREISFÖRMIG)

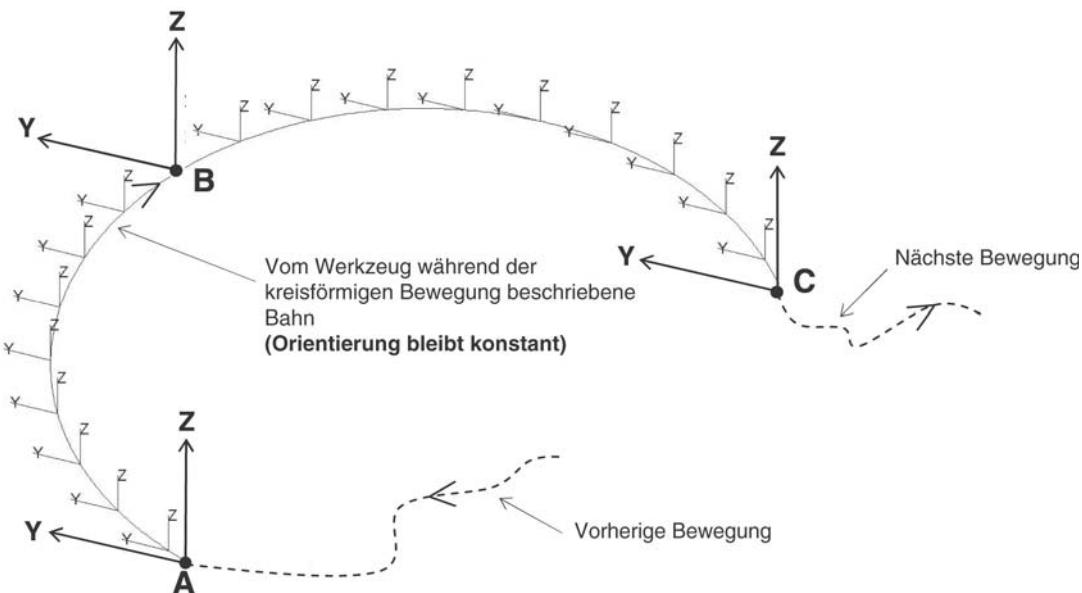
10.1.4.1. INTERPOLATION DER ORIENTIERUNG

Um die Orientierung zu ändern, minimiert die **VAL 3**-Bahnplanung stets die Amplitude der Werkzeugdrehungen.

Dadurch kann als Sonderfall, für alle geradlinigen oder kreisförmigen Bewegungen, eine absolute, konstante Orientierung oder eine Orientierung bezüglich der Bahn programmiert werden.

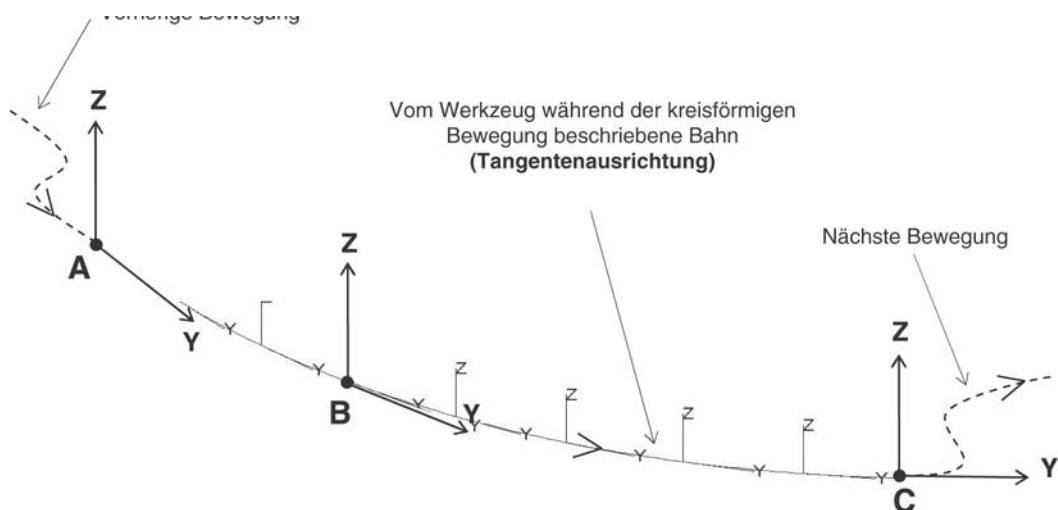
- Für eine konstante Orientierung müssen Ausgangs- und Endposition sowie die Mittelposition für einen Kreis dieselbe Orientierung haben.

Absolute, konstante Ausrichtung



- Für eine konstante Orientierung bezüglich der Bahn (z. B. Richtung Y der Werkzeugmarkierung Bahntangente) müssen Ausgangs- und Endposition sowie die Mittelposition für einen Kreis dieselbe Orientierung haben.

Konstante Orientierung bezüglich der Bahn

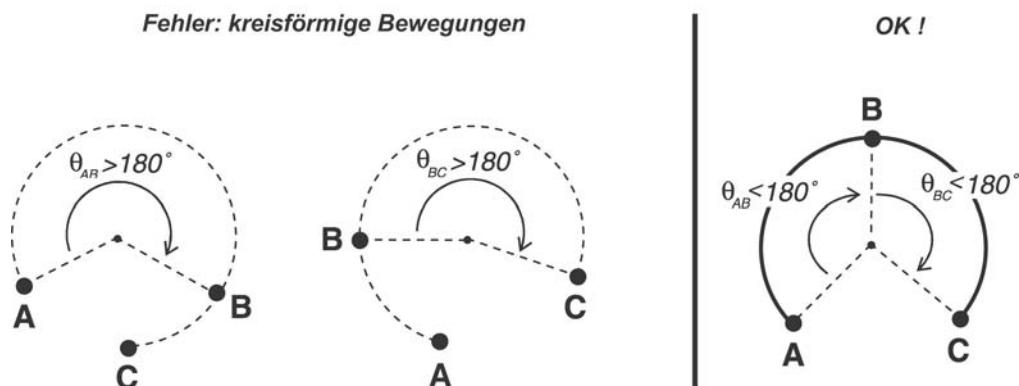


Daraus ergibt sich eine Beschränkung für die kreisförmigen Bewegungen:

Wenn der mittlere Punkt mit dem Anfangs- oder mit dem Endpunkt einen Winkel von **180°** oder mehr bildet, gibt es mehrere Interpolations- und Orientierungsmöglichkeiten und es wird ein Fehler erzeugt.

Die Position des mittleren Punktes muss also geändert werden, um die Mehrdeutigkeit der mittleren Orientierung zu beenden.

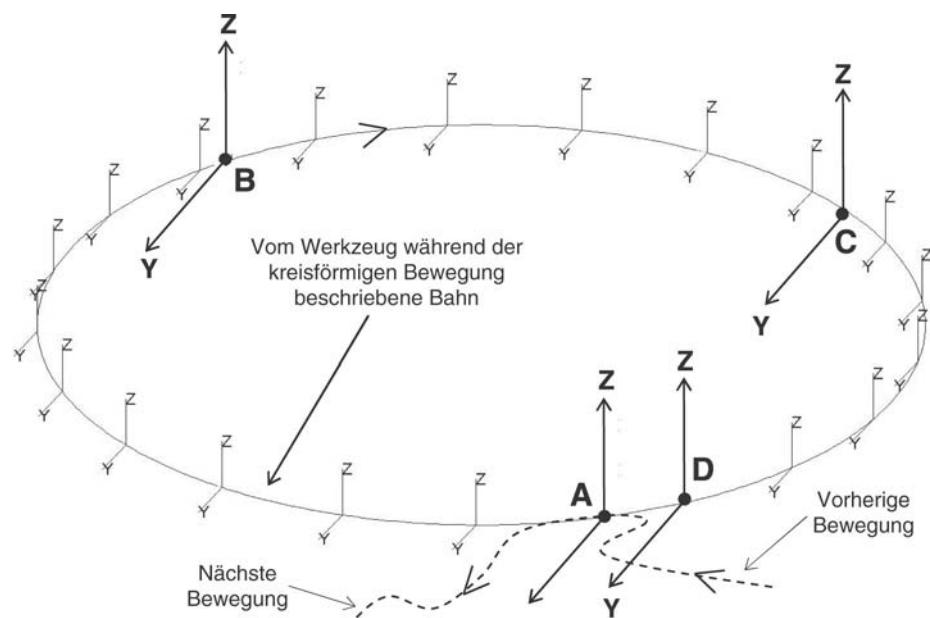
Mehrdeutigkeit der mittleren Orientierung



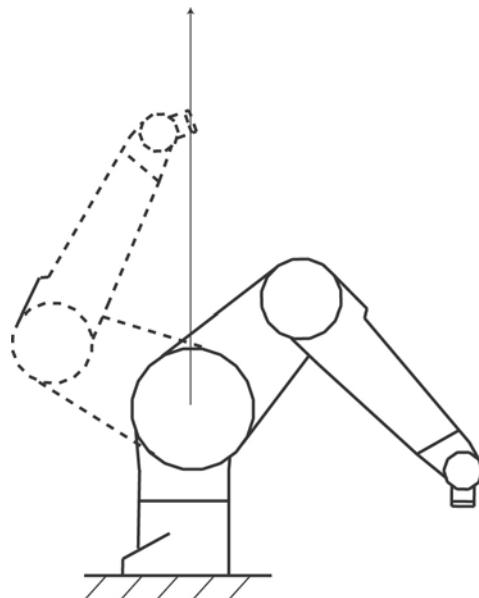
Insbesondere verlangt die Programmierung eines vollständigen Kreises **2 movec-Anweisungen**:

```
movec (B, C, tTool, mDesc)
movec (D, A, tTool, mDesc)
```

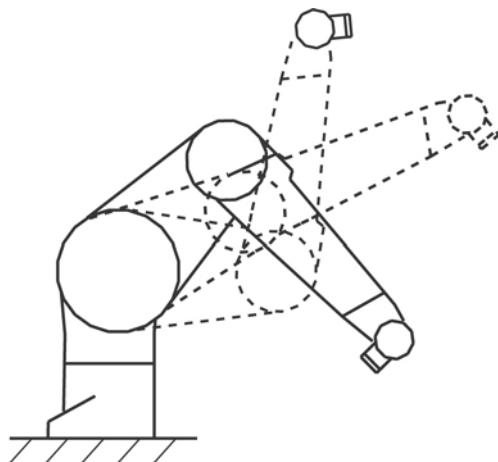
Vollständiger Kreis



10.1.4.2. ÄNDERUNG DER KONFIGURATION (ARM RX/TX)

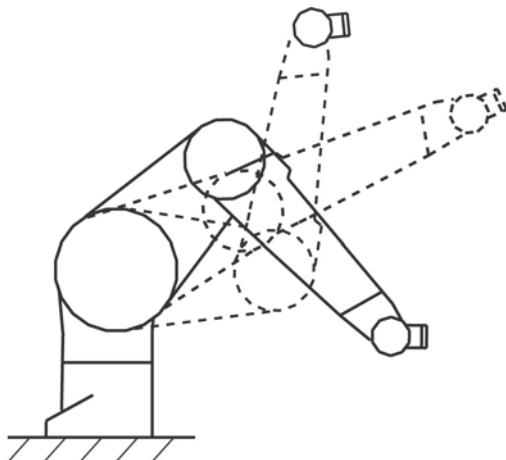
Wechsel: righty / lefty

Bei einer Änderung der Schulterkonfiguration muss der Mittelpunkt des Handgelenks über die Achse 1 hinwegfahren (nicht unbedingt bei Robotern mit Ausleger).

Positive / negative Konfiguration des Ellenbogens

Bei Änderung der Konfiguration des Ellenbogens muss der Arm gestreckt ($j_3 = 0^\circ$) werden.

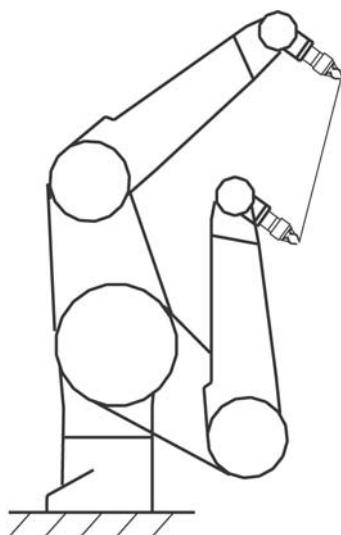
Positive / negative Konfiguration des Handgelenks



Bei einer Änderung der Konfiguration des Handgelenks muss der Arm kurzzeitig das Handgelenk ($j5 = 0^\circ$) strecken.

Das bedeutet, dass der Roboter auf seinem Weg bestimmte Positionen einnehmen muss, wenn die Konfiguration geändert wird. Man kann aber keine geradlinigen oder kreisförmigen Bewegungen über diese Positionen erzwingen, wenn sie nicht auf der gewünschten Bahn liegen! Daraus folgt, **dass ein Konfigurationswechsel während einer geradlinigen oder kreisförmigen Bewegung nicht erzwungen werden kann.**

Wechseln der Ellenbogenkonfiguration nicht möglich

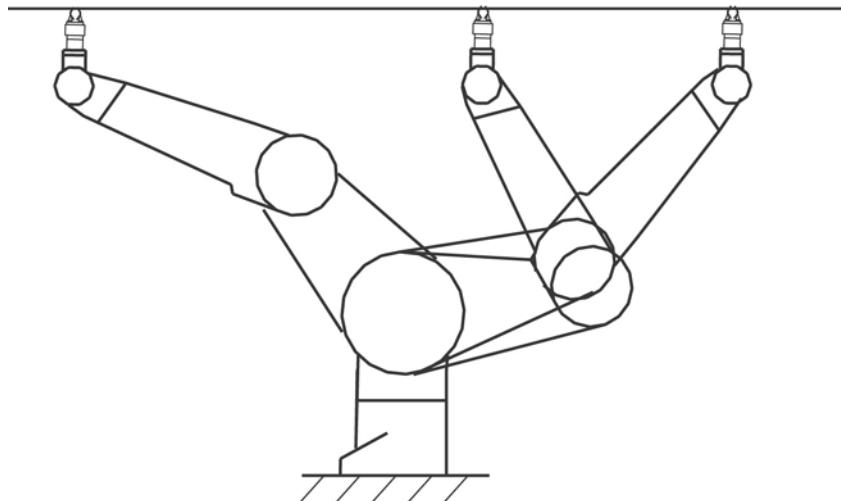


Anders ausgedrückt kann man bei einer geradlinigen oder kreisförmigen Bewegung eine Konfiguration nur dann realisieren, wenn sie mit der Anfangsposition vereinbar ist: Eine freie oder mit der Ausgangsposition identische Konfiguration kann immer spezifiziert werden.

Im Ausnahmefall kann die Gerade oder der Kreisbogen durch eine Position gehen, an der ein Konfigurationswechsel möglich ist. In diesem Fall kann das System, wenn die Konfiguration nicht vorher festgelegt wurde, die Konfiguration auf einer geradlinigen oder kreisförmigen Bewegung ändern.

Bei einer kreisförmigen Bewegung wird die Konfiguration des mittleren Punktes nicht berücksichtigt. Es zählen nur die Konfigurationen von Anfangs- und Endposition.

Andere Schulterkonfiguration möglich



10.1.4.3. SINGULARITÄTEN (ARM RX/TX)

Singularitäten sind ein charakteristisches Merkmal aller 6-Achsen-Roboter. Sie können als Punkte definiert werden, an denen der Roboter die Konfiguration ändert. Bei einer Fluchtung zweier Achsen bilden diese eine einzige Achse: Der 6-Achsen-Roboter verhält sich lokal wie ein 5-Achsen-Roboter. Damit können bestimmte Bewegungen nicht mehr ausgeführt werden. Das stört bei einer Punktsteuerung nicht: Die vom System generierten Bewegungen sind immer noch möglich. Bei einer geradlinigen oder kreisförmigen Bewegung wird jedoch die Bahn dieser Bewegung vorgeschrieben. Wenn die Bewegung nicht möglich ist, wird während der Bewegung eine Fehlermeldung erzeugt.

10.2. ANTICIPATION VON BEWEGUNGEN

10.2.1. PRINZIP

Das System steuert die Bewegungen des Roboters etwa wie ein Autofahrer sein Auto. Es passt die Geschwindigkeit des Roboters an die Bahngeometrie an. Je früher die Bahn bekannt ist, desto optimaler kann das System die Geschwindigkeit steuern. Deshalb wartet das System zur Verarbeitung der nächsten Bewegungsanweisungen nicht darauf, dass die laufende Bewegung zu Ende geführt wird.

Betrachten wir die folgenden Programmzeilen:

```
movej(pA, tTool, mDesc)
movej(pB, tTool, mDesc)
movej(pC, tTool, mDesc)
movej(pD, tTool, mDesc)
```

Es wird vorausgesetzt, dass der Roboter bei Erreichen dieser Programmzeilen stillsteht. Wenn die erste Anweisung ausgeführt ist, beginnt der Roboter mit der Bewegung zum Punkt **pA**. Das Programm setzt jedoch sofort mit der zweiten Zeile fort, d. h. lange bevor der Roboter den Punkt **pA** erreicht.

Wenn das System die zweite Zeile ausführt, beginnt der Roboter erst mit seiner Bewegung nach **pA** und die Information, dass der Roboter nach dem Punkt **pA** zum Punkt **pB** fahren soll, wird vom System gespeichert. Das Programm setzt mit der nächsten Zeile fort: Während der Roboter immer noch auf dem Weg nach **pA** ist, speichert das System bereits die Bewegung von **pB** nach **pC**. Da das Programm sehr viel schneller läuft als sich der Roboter bewegt, ist dieser wahrscheinlich immer noch auf dem Weg nach **pA**, wenn die nächste Zeile abgearbeitet wird. Auf diese Weise werden vom System mehrere der nächsten Punkte gespeichert.

So weiß der Roboter bereits zu dem Zeitpunkt, wo er seine Bewegung zu **pA** beginnt, dass er nach **pA** zu **pB**, dann zu **pC** und schließlich zu **pD** fahren muss. Wenn die Funktion Glättung aktiv ist, weiß das System, dass der Roboter erst am Punkt **pD** anhalten muss. Es kann daher stärker beschleunigen, als wenn es sich darauf vorbereiten müsste, in **pB** oder **pC** anzuhalten.

Das Ausführen der Programmzeilen führt nur zur Speicherung der aufeinanderfolgenden Bewegungsanweisungen. Der Roboter realisiert sie anschließend nach seinen Möglichkeiten. Damit das System eine möglichst optimale Bahn wählen kann, muss der Speicher, in dem die Bewegungen abgelegt werden, relativ groß sein. Die Speicherkapazität ist jedoch begrenzt. Wenn der Speicher voll ist, wird die Programmausführung bei der nächsten Bewegungsanweisung unterbrochen. Die Ausführung wird erst fortgesetzt, wenn der Roboter die laufende Bewegung beendet hat und der entsprechende Speicherplatz frei geworden ist.

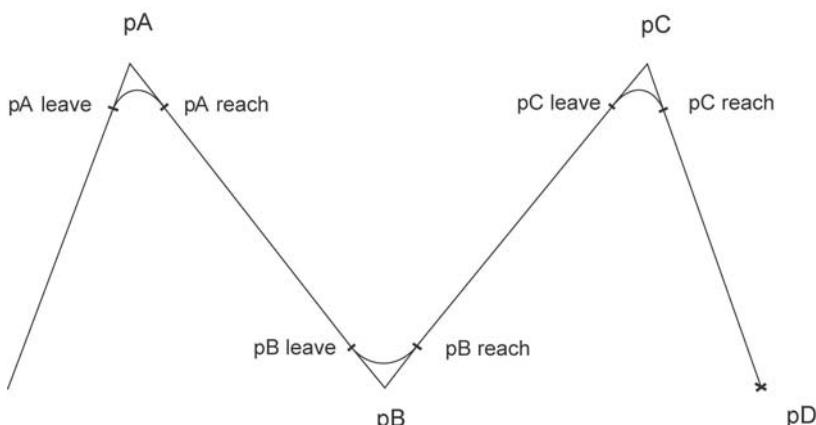
10.2.2. ANTICIPATION UND BLENDING

Wir sehen in diesem Kapitel, was im Einzelnen passiert, wenn die Bewegungen verkettet sind. Betrachten wir wieder das obige Beispiel:

```
movej(pA, tTool, mDesc)
movej(pB, tTool, mDesc)
movej(pC, tTool, mDesc)
movej(pD, tTool, mDesc)
```

Angenommen, im Bewegungsdeskriptor **mDesc** ist das Blending aktiviert. Wenn die erste Zeile ausgeführt wird, weiß das System noch nicht, welches die nächste Bewegung sein wird. Nur die Bewegung zwischen dem Startpunkt und dem Punkt **pA leave** ist vollständig bestimmt, da der Punkt **pA leave** vom System anhand des Datenelements **leave** des Bewegungsdeskriptors ermittelt wird (siehe nachstehendes Schema).

Zyklus mit Blending



Solange die zweite Zeile nicht ausgeführt ist, kann Bahnabschnitt mit Blending um den Punkt **pA** nicht vollständig bestimmt werden, da das System die nächste Bewegung noch nicht kennt. Im Schrittmodus würde der Roboter, wenn der Benutzer nicht auf die nächste Anweisung weitergeht, nur bis zum Punkt **pA leave** fahren. Wenn die nächste Anweisung ausgeführt wurde, können die geglättete Bahn um den Punkt **pA** (zwischen **pA leave** und **pA reach**) sowie die Bewegung zum Punkt **pB leave** definiert werden. Der Roboter kann sich damit bis **pB leave** bewegen. Er wird aber im Schrittmodus nicht über diesen Punkt hinausfahren können, wenn der Benutzer die dritte Anweisung noch nicht ausgeführt hat usw.

Der Vorteil dieser Betriebsart ist, dass der Roboter im Schrittmodus und bei normaler Programmausführung genau die gleiche Bewegung ausführt.

10.2.3. SYNCHRONISIERUNG

Das Antizipieren besteht also in einer zeitlichen Entkopplung der Ausführung der Programmzeilen unter **VAL 3** und den entsprechenden Bewegungen des Roboters: Das **VAL 3**-Programm eilt dem Roboter voraus.

Wenn an einer bestimmten Position des Roboters eine Aufgabe vorgesehen ist, muss das Programm warten, bis der Roboter seine Bewegungen beendet hat: Diese Synchronisierung wird mit der Anweisung **waitEndMove()** erreicht. Eine Alternative besteht darin, die Anweisung **getMoveld()** zu nutzen, um das Fortschreiten des Arms auf der Bahn zu erkennen (siehe 10.4, Bewegungskontrolle in Echtzeit).

Deshalb wird im nachstehenden Programm:

```
movej(A, tTool, mDesc)
movej(B, tTool, mDesc)
waitEndMove()
movej(C, tTool, mDesc)
movej(D, tTool, mDesc)
etc...
```

Die beiden ersten Zeilen werden ausgeführt, wenn der Roboter seinen Weg nach **A** beginnt. Dann wird die Programmausführung in der dritten Zeile unterbrochen, bis der Roboter im Punkt **B** steht. Nach der Stabilisierung in **B** setzt das Programm mit der Ausführung fort.

Die Anweisungen **open()** und **close()** bewirken ebenfalls ein Warten auf das Ende der Roboterbewegung bevor das Werkzeug betätigt wird.

10.3. GESCHWINDIGKEITSSTEUERUNG

10.3.1. PRINZIP

Die Geschwindigkeitssteuerung auf einer Bahn arbeitet nach folgendem Prinzip:

Zu jedem Zeitpunkt wird der Roboter mit voller Leistung bewegt und beschleunigt, wobei alle von der Bewegungssteuerung vorgegebenen Bedingungen für Geschwindigkeit und Beschleunigung einzuhalten sind.

Die Bewegungsanweisungen enthalten zwei Arten von Randbedingungen für die Geschwindigkeit, die in einer Variablen des Typs **mdesc** definiert sind:

1. Geschwindigkeits- (Achsgeschwindigkeiten), Beschleunigungs- und Abbremsungseinschränkungen
2. Randbedingungen für die kartesischen Geschwindigkeiten des Werkzeugmittelpunkts

Der Beschleunigungswert legt fest, wie rasch die Geschwindigkeit am Beginn der Bahn zunimmt. Umgekehrt wird durch den Wert der Abbremsung festgelegt, wie schnell die Geschwindigkeit am Ende der Bahn abnimmt. Wenn man hohe Beschleunigungs- und Abremswerte verwendet, werden die Bewegungen zwar rascher, aber ruckartiger. Mit kleineren Werten nehmen die Bewegungen etwas mehr Zeit in Anspruch, werden jedoch sanfter.

10.3.2. EINFACHE EINSTELLUNG

Wenn das Werkzeug oder der vom Roboter transportierte Gegenstand keine besonders vorsichtige Handhabung erfordern, sind Randbedingungen für kartesische Geschwindigkeiten unnötig. Die Einstellung der Bahngeschwindigkeit erfolgt im Allgemeinen auf folgende Weise:

1. Die Randbedingungen für kartesische Geschwindigkeit werden auf hohe Werte gesetzt, z.B. auf die vorprogrammierten Werte, damit sie keinen Einfluss auf die anschließende Einstellung haben.
2. Geschwindigkeit, Beschleunigungen und Abbremsungen der Gelenke werden auf ihre Nennwerte (**100%**) initialisiert.
3. Dann wird die Bahngeschwindigkeit nur mit dem Parameter vel eingestellt.

Um ein harmonisches Verhalten des Arms zu bewahren, müssen Beschleunigung und Abbremsung mit der Geschwindigkeit geändert werden: Die Parameter Beschleunigung und Verzögerung sollten ungefähr das Quadrat des Parameters Geschwindigkeit betragen. Zum Beispiel passt eine Geschwindigkeit von 120 % = 1.2 am besten zu einer Beschleunigung und Verzögerung von $1.2 \times 1.2 = 1.44 = 144\text{ %}$. Höhere Werte für Beschleunigung und Verzögerung führen zu einem aggressiveren aber unsauberen Verhalten des Arms.

10.3.3. KOMPLEXERE EINSTELLUNGEN

Wenn die kartesische Geschwindigkeit des Werkzeugs beherrscht werden muss, zum Beispiel um eine Bahn mit konstanter Geschwindigkeit auszuführen, sollte besser auf folgende Weise vorgegangen werden:

1. Die Bedingungen für die kartesischen Geschwindigkeiten auf die zunächst gewünschten Werte einstellen.
2. Geschwindigkeit, Beschleunigungen und Abbremsungen der Gelenke werden auf ihre Nennwerte (**100%**) initialisiert.
3. Dann die Bahngeschwindigkeit nur mit den Parametern für kartesische Geschwindigkeit einstellen.
4. Wenn die gewünschte Geschwindigkeit nicht erreicht werden kann, müssen die Beschleunigungs- und Abbremswerte vergrößert werden.

Wenn man in starken Kurven automatisch bremsen möchte, müssen die Beschleunigungs- und Abbremswerte verringert werden.

10.3.4. SCHLEPPFEHLER

Die Nennwerte für die Achsgeschwindigkeiten und Beschleunigungen gelten für Roboter mit Nennlast unabhängig von der Bahn.

In vielen Fällen kann der Roboter schnellere Bewegungen ausführen: Seine Höchstgeschwindigkeiten hängen jedoch von der Last und der Bahn ab. Unter günstigen Bedingungen (geringe Last, günstiger Einfluss der Schwerkraft) kann der Roboter seine Nennwerte ohne Schaden überschreiten.

Wenn der Roboter hingegen eine schwerere Last bewegen muss als die Nennlast, oder wenn zu hohe Achsgeschwindigkeiten und Beschleunigungen parametriert wurden, kann der Roboter unter Umständen den Bewegungsanweisungen nicht mehr folgen und bleibt mit einem Schleppfehler stehen. Durch Eingabe kleinerer Geschwindigkeits- und Beschleunigungswerte können derartige Fehler vermieden werden.

ACHTUNG:

Bei geradlinigen Bewegungen in der Nähe singulärer Punkte sind große Achsbewegungen für kleine Werkzeugbewegungen erforderlich. Wenn eine zu hohe Geschwindigkeit eingegeben wurde, wird der Roboter der Anweisung nicht mehr folgen können und mit einem Schleppfehler stehen bleiben.

10.4. ONLINE-BEWEGBUNGSTEUERUNG

Die bis jetzt angesprochenen Bewegungssteuerungen haben keine sofortige Wirkung: Die Ausführung einer solchen Steueranweisung vom Programm führt zur Speicherung der Bewegungsanweisung durch das System. Sie werden erst anschließend vom Roboter ausgeführt.

Es besteht jedoch die Möglichkeit, die Bewegungen des Roboters mit sofortiger Wirkung zu beeinflussen:

- Die Monitorgeschwindigkeit ändert die Geschwindigkeiten aller Bewegungen. Sie kann mit sofortiger Wirkung mit der Anweisung `setMonitorSpeed()` geändert werden. Allerdings kann diese Anweisung die Geschwindigkeit nicht erhöhen, wenn der Bediener sie ebenfalls am MCP anpassen kann.
- Mit den Anweisungen `stopMove()` und `restartMove()` kann die Bewegung auf der Bahn unterbrochen und wieder aufgenommen werden.
- Die Anweisung `resetMotion()` ermöglicht die Unterbrechung einer Bewegung und das Löschen der gespeicherten Bewegungsanweisungen.
- Die Anweisung Alter (Option) bezieht sich auf die unmittelbar effektive Bahn einer Koordinatentransformation (Translation, Rotation, Rotation am Werkzeugnullpunkt).
- Es ist möglich, die Position des Roboters genau und in Echtzeit auf der Bahn mit der `getMoveld()`-Anweisung zu verfolgen. Jeder Fahrbefehl wird mit einem numerischen Wert identifiziert, der von der Anweisung zurückgegeben wird. Die `getMoveld()`-Anweisung liefert einen Zahlenwert der die aktuelle Bewegung (ganzzahliger Teil) und das Fortschreiten der Bewegung (Dezimalteil) identifiziert. Beispielsweise bedeutet eine Bewegungs-ID von 17.572, dass die aktuelle Bewegung die Bewegungsanweisung ist, die 17 liefert und die Roboterposition 57.2 % dieser Bewegung entspricht.

10.5. TYP MDESC

10.5.1. DEFINITION

Mit dem Typ **mdesc** können die Parameter einer Bewegung festgelegt werden (Geschwindigkeit, Beschleunigung, Glättung).

Der Typ **mdesc** ist ein strukturierter Typ mit folgenden Feldern in der nachstehenden Reihenfolge:

| | |
|--------------------|---|
| num accel | Maximal zulässige Achsbeschleunigung, angegeben in % der Nennbeschleunigung des Roboters. |
| num vel | Maximal zulässige Achsgeschwindigkeit, angegeben in % der Nenngeschwindigkeit des Roboters. |
| num decel | Maximal zulässige Achsverzögerung, angegeben in % der nominalen Verzögerung des Roboters. |
| num tvel | Maximal zulässige Lineargeschwindigkeit des Werkzeugmittelpunkts in mm/s oder inch/s je nach Längeneinheit des Programms. |
| num rvel | Maximal zulässige Winkelgeschwindigkeit des Werkzeugs in Grad/s. |
| blend blend | Glättungsart: off (kein Blending) joint oder Cartesian (mit Blending). |
| num leave | Bei joint und Cartesian Blending, die Entfernung vom Zielpunkt bei der das Blending startet und dem nächsten Punkt, angegeben in mm oder inch, je nach Längeneinheit der Applikation. |
| num reach | Bei joint und Cartesian Blending, die Entfernung vom Zielpunkt bei der das Blending endet und dem nächsten Punkt, angegeben in mm oder inch, je nach Längeneinheit der Applikation. |

Ausführliche Beschreibung der verschiedenen Parameter, siehe Kapitel Bewegungssteuerung.

Der vorprogrammierte Wert für eine Variable des Typs **mdesc** ist **{100,100,100,9999,9999,joint,50,50}**.

10.5.2. OPERATOREN

Mit zunehmender Priorität:

| | |
|---|---|
| mdesc <mdesc& desc1> = <mdesc desc2> | Weist jedes Feld desc2 dem entsprechenden Feld der Variablen desc1 zu. |
| bool <mdesc desc1> != <mdesc desc2> | Überträgt true , wenn sich desc1 und desc2 in mindestens einem Feld unterscheiden. |
| bool <mdesc desc1> == <mdesc desc2> | Überträgt true , wenn die Werte der Felder in desc1 und desc2 gleich sind. |

10.6. BEWEGUNGSANWEISUNGEN

num movej(joint jPosition, tool tTool, mdesc mDesc)

num movej(point pPosition, tool tTool, mdesc mDesc)

Funktion

Diese Anweisung speichert einen Bewegungsbefehl für eine Joint-Bewegung zur Position **pPosition** oder **jPosition** mit dem Werkzeug **tTool** und dem Motiondescriptor **mDesc**. Sie liefert die dieser Bewegung zugewiesene Bewegungs-ID zurück und erhöht die Bewegungs-ID für den nächsten Bewegungsbefehl um eins.

ACHTUNG:

Das System wartet nicht auf das Ende der Bewegung, um zur nächsten VAL 3-Anweisung weiterzugehen: Es können mehrere Bewegungsanweisungen vorab gespeichert werden. Wenn nicht mehr genügend Speicherplatz für eine neue Anweisung verfügbar ist, wird mit der Ausführung gewartet, bis die Speicherung möglich ist.

Zur ausführlichen Beschreibung der verschiedenen Bewegungsparameter, siehe Beginn des Kapitels Bewegungssteuerung.

Wenn **mDesc** unzulässige Werte aufweist, **jPosition** außerhalb der Softwaregrenzen liegt, **pPosition** nicht erreichbar ist oder ein zuvor aufgezeichneter Bewegungsbefehl nicht ausgeführt werden kann, wird ein Laufzeitfehler erzeugt (Endpunkt nicht erreichbar).

Der erste Parameter der Funktion movej kann entweder ein joint (Gelenktyp) oder ein point (Punkttyp) Wert sein. Aufgrund einer internen Begrenzung ist der Compiler nicht in der Lage, den Eingangsdatentyp zu prüfen, wenn ein expliziter Ausdruck für diesen Parameter eingegeben wird.

Beispiel

```
movej({0,0,0,0,0,0},flange,mNomSpeed)
→ "{0,0,0,0,0,0}": Ambiguous expression
The above expression cannot be used and must be replaced by the following one...
myJoint={0,0,0,0,0,0}
movej(myJoint,flange,mNomSpeed)
```

Siehe auch

```
num movel(point pPosition, tool tTool, mdesc mDesc)
bool isInRange(joint jPosition)
void waitEndMove()
num movec(point plntermediate, point pTarget, tool tTool, mdesc mDesc)
```

num movel(point pPosition, tool tTool, mdesc mDesc)

Funktion

Diese Anweisung speichert einen linearen Bewegungsbefehl zum Punkt **pPosition** mit dem Werkzeug **tTool** und dem Motiondescriptor **mDesc**. Sie liefert die dieser Bewegung zugewiesene Bewegungs-ID zurück und erhöht die Bewegungs-ID für den nächsten Bewegungsbefehl um eins.

ACHTUNG:

Das System wartet nicht auf das Ende der Bewegung, um zur nächsten VAL 3-Anweisung weiterzugehen: Es können mehrere Bewegungsanweisungen vorab gespeichert werden. Wenn nicht mehr genügend Speicherplatz für eine neue Anweisung verfügbar ist, wird mit der Ausführung gewartet, bis die Speicherung möglich ist.

Zur ausführlichen Beschreibung der verschiedenen Bewegungsparameter, siehe Beginn des Kapitels Bewegungssteuerung.

Wenn **mDesc** unzulässige Werte aufweist, **pPosition** nicht erreichbar ist, eine geradlinige Bewegung zu **pPosition** nicht möglich ist oder ein zuvor aufgezeichneter Bewegungsbefehl nicht ausgeführt werden kann, wird ein Laufzeitfehler erzeugt (Endpunkt nicht erreichbar).

Siehe auch

num movej(joint jPosition, tool tTool, mdesc mDesc)
void waitEndMove()
num movec(point plntermediate, point pTarget, tool tTool, mdesc mDesc)

```
num movec(point pIntermediate, point pTarget, tool tTool,
          mdesc mDesc)
```

Funktion

Diese Anweisung speichert die Anweisung für eine kreisförmige Bewegung, ausgehend vom Endpunkt der vorherigen Bewegung, über point **pIntermediate** bis zu point **pTarget**. Sie liefert die dieser Bewegung zugewiesene Bewegungs-ID zurück und erhöht die Bewegungs-ID für den nächsten Bewegungsbefehl um eins.

Die Ausrichtung des Werkzeugs ist so interpoliert, dass eine absolute, konstante Ausrichtung oder eine Ausrichtung bezüglich der Bahn möglich ist.

ACHTUNG:

Das System wartet nicht auf das Ende der Bewegung, um zur nächsten VAL 3-Anweisung weiterzugehen: Es können mehrere Bewegungsanweisungen vorab gespeichert werden. Wenn nicht mehr genügend Speicherplatz für eine neue Anweisung verfügbar ist, wird mit der Ausführung gewartet, bis die Speicherung möglich ist.

Zur ausführlichen Beschreibung der verschiedenen Bewegungsparameter und der Interpolation der Ausrichtung, siehe Beginn des Kapitels "Bewegungssteuerung".

Bei ungültigen Werten von **mDesc** wird ein Laufzeitfehler erzeugt. Dies gilt auch, wenn die Positionen point **pIntermediate** (oder point **pTarget**) nicht erreichbar sind, wenn die kreisförmige Bewegung nicht möglich ist (siehe Kapitel "Bewegungssteuerung" - Interpolation der Ausrichtung) oder wenn ein zuvor aufgezeichneter Bewegungsbefehl nicht ausgeführt werden kann (Zielposition nicht erreichbar).

Siehe auch

num movej(joint jPosition, tool tTool, mdesc mDesc)
num movel(point pPosition, tool tTool, mdesc mDesc)
void waitEndMove()

void stopMove()

Funktion

Diese Anweisung hält den Arm auf der Bahn an und entzieht die Berechtigung für programmierte Bewegungen.

ACHTUNG:

Diese Anweisung bewirkt sofortige Rückkehr, die VAL 3-Task wartet zur Ausführung der folgenden Task nicht auf den Stillstand des Arms.

Der Motiondescriptor zur Ausführung des Stopps ist derjenige, der auch zur Ausführung der aktuellen Bewegung genutzt wurde.

Die Bewegung kann erst nach Ausführung der Anweisung **restartMove()** oder **resetMotion()** fortgesetzt werden.

Nicht programmierte Bewegungen (manuelles Verfahren) bleiben weiterhin möglich.

Beispiel

```
// waits for a signal
wait(diSignal==true)
// stops movements along the trajectory
stopMove()
wait(diSignal==false)
// restarts movements along the trajectory
restartMove()
```

Siehe auch

void restartMove()
void resetMotion(), void resetMotion(joint jStartingPoint)

void resetMotion(), void resetMotion(joint jStartingPoint)

Funktion

Diese Anweisung hält den Roboterarm während der Bewegung an und löscht alle gespeicherten Bewegungsanweisungen. Sie setzt die Bewegungs-ID auf Null.

ACHTUNG:

Diese Anweisung bewirkt sofortige Rückkehr, die VAL 3-Task wartet zur Ausführung der folgenden Task nicht auf den Stillstand des Arms.

Wenn die programmierte Freigabe der Bewegung durch die Anweisung **stopMove()** aufgehoben worden war, wird sie wieder hergestellt.

Wenn die **jStartingPoint** Joint-Position festgelegt ist, kann die nächste Bewegungsanweisung nur von dieser Position aus durchgeführt werden: Es ist zunächst eine Anschluss-Bewegung erforderlich, um zu **jStartingPoint** zurückzukehren.

Ist keine Joint-Position festgelegt, so wird die nächste Bewegungsanweisung von der aktuellen Position des Arms ausgeführt, egal, wo sich dieser befindet.

Siehe auch

bool isEmpty()
void stopMove()
void autoConnectMove(bool bActive), bool autoConnectMove()
num setMovId(num nMovId)
joint resetTurn(joint jReference)

void restartMove()

Funktion

Diese Anweisung gibt die Berechtigung für programmierte Bewegungen und setzt die mit der Anweisung **stopMove()** unterbrochene Bewegung fort.

Wenn die programmierte Bewegungsfreigabe nicht durch **stopMove()** unterbrochen war, hat diese Anweisung keinerlei Auswirkungen.

Siehe auch

void stopMove()
void resetMotion(), void resetMotion(joint jStartingPoint)

void waitEndMove()

Funktion

Diese Anweisung annulliert das Blending der zuletzt gespeicherten Bewegungsanweisung und wartet auf die Ausführung dieser Anweisung.

Diese Anweisung wartet nicht auf die Stabilisierung des Roboters an seiner Endposition, sondern nur dass die an die Verstärker gesendete Positionsanweisung mit der gewünschten Endposition übereinstimmt. Wenn die vollständige Stabilisierung nach der Bewegung abgewartet werden soll, so muss die Anweisung **isSettled()** verwendet werden.

Ein Laufzeitfehler wird erzeugt, wenn ein zuvor aufgezeichneter Bewegungsbefehl nicht ausgeführt werden kann (Endpunkt außer Reichweite).

Beispiel

(Siehe Kapitel 10.2)

Siehe auch

bool isSettled(), bool isSettled(tool tTool, num nTransAccuracy), bool isSettled(tool tTool, num nTransAccuracy, num nRotAccuracy, num nTime)
bool isEmpty()
void stopMove()
void resetMotion(), void resetMotion(joint jStartingPoint)

bool isEmpty()

Funktion

Diese Anweisung liefert **true** zurück, wenn alle Bewegungsanweisungen ausgeführt sind, und **false**, wenn noch mindestens eine Anweisung ansteht.

Beispiel

Dieses Programm löscht gespeicherte Bewegungen, wenn es welche gibt:

```
// If commands are in progress
if isEmpty()==false
// Stop the robot and cancel the commands
resetMotion()
sOutput="Movements have been cancelled"
endif
```

Siehe auch

void waitEndMove()
void resetMotion(joint jStartingPoint)

bool isSettled(), bool isSettled(tool tTool, num nTransAccuracy),
bool isSettled(tool tTool, num nTransAccuracy, num nRotAccuracy,
num nTime)

Funktion

Diese Anweisung überträgt **true**, wenn der Roboter gestoppt ist, und **false**, wenn seine Position noch nicht stabilisiert ist.

ACHTUNG:

Der Roboter kann aus verschiedenen Gründen gestoppt werden und deswegen auch stabilisiert sein, bevor alle gespeicherten Bewegungen ausgeführt sind. Verwenden Sie isEmpty(), um festzustellen, ob der Roboter am Ende seiner programmierten Bewegung gestoppt wurde.

Wenn kein Argument vorhanden ist, gilt die Position als stabilisiert, wenn der Positionsfehler für jedes Gelenk kleiner als 1% des maximal zulässigen Gelenkpositionsfehlers in jeder Standardstabilisierungszeit des Arms bleibt (die Hälfte der Periode des normalen Oszillationsmodus, der vom Armmodell abhängt).

Mit 2 Argumenten (Werkzeug- und Translationspräzision) gilt die Position als stabilisiert, wenn die kartesische Position der Werkzeugspitze innerhalb der angegebenen Präzision für die Standardstabilisierungszeit des Arms liegt.

Mit 4 Argumenten (Werkzeug-, Translations- und Rotationspräzision und Zeit) gilt die Position als stabilisiert, wenn die kartesische Position der Werkzeugspitze innerhalb der angegebenen Präzision für die angegebene Zeit bleibt.

Siehe auch

bool isEmpty()
void waitEndMove()

void autoConnectMove(bool bActive), bool autoConnectMove()

Funktion

Im ferngesteuerten Modus ist die Anschlussbewegung automatisch, wenn der Arm sich sehr nahe an seiner Bahn bewegt (Abstand kleiner als der maximal erlaubte Schleppfehler). Ist der Arm zu weit von seiner Bahn entfernt, findet die Anschlussbewegung automatisch oder unter manueller Kontrolle gemäß dem in der Anweisung **autoConnectMove** definierten Modus statt.: automatisch, wenn **bActive** den Wert **true** hat, unter manueller Kontrolle, wenn der Wert **bActive** gleich **false** ist. Bei Benutzung ohne Parameter, gibt **autoConnectMove** den üblichen Modus der Anschlussbewegung an.

Standardmäßig erfolgt die Anschlussbewegung bei ferngesteuerten Betrieb manuell.

ACHTUNG:

Unter normalen Einsatzbedingungen stoppt der Arm bei einem Notaus auf seiner Bahn. Demzufolge kann der Arm im ferngesteuerten Modus unabhängig von dem in der Anweisung **autoConnectMove definierten Modus der Anschlussbewegung automatisch neu starten.**

Siehe auch

void resetMotion(), void resetMotion(joint jStartingPoint)

num getSpeed(tool tTool)

Funktion

Diese Anweisung liefert die aktuelle kartesische Geschwindigkeit am TCPtTool des spezifizierten Werkzeugs **tTool**. Die Geschwindigkeit wird aus den Sollgeschwindigkeiten und nicht aus den Istgeschwindigkeiten der Achsen berechnet.

Siehe auch

point here(tool tTool, frame fReference)

joint getPositionErr()

Funktion

Diese Anweisung gibt den aktuellen Schleppfehler der Achsen an. Der Schleppfehler ist der Unterschied zwischen der an die Antriebe gesendeten Sollposition und dem von den Wegmesssystemen der Achsen zurückgelieferten Wert.

Siehe auch

void getJointForce(num& nForce)

void getJointForce(num& nForce)

Funktion

Diese Anweisung liefert das aktuelle Drehmoment (N.m der Drehachse) oder die Kraft (N für die Linearachse), die aus den Motorströmen berechnet werden.

Die Gelenkkraft ist keine Schätzung ausschließlich externer Kräfte. Sie beinhaltet auch Schwerkraft, Reibung, Trägheit, Rauschen und Auflösung der Stromsensoren sowie das Verhältnis zwischen Motorstrom und Drehmoment. Sie kann nur durch Aufzeichnen von Kräften unter Referenzbedingungen und deren Vergleich mit unter ähnlichen Bedingungen und externer Sensorik gemessenen Kräften zur Bewertung herangezogen werden.

Der Rückgabewert ist nur ein Maß für die Größenordnung der Kräfte. Die Genauigkeit der Rückgabewerte kann nicht garantiert werden; sie muss für jede Applikation neu bewertet werden.

Ist der Parameter kein num-Array ausreichender Größe, so wird ein Laufzeitfehler erzeugt.

Siehe auch**joint getPositionErr()****num getMoveld()**

Funktion

Diese Anweisung liefert einen numerischen Wert, der die aktuelle Position des Roboters auf der Bahn angibt. Der ganzzahlige Teil verweist auf die Nummer der Bewegungsanweisung, die ausgeführt wird. Diese Ganzzahl wird mit der Ausführung der Bewegungsanweisung geliefert. Der Dezimalteil zeigt das Fortschreiten in % dieser Bewegung an.

Eine Bewegungs-ID sollte niemals mit dem Operator '==' getestet werden, sondern mit dem Operator '>=': `wait(getMoveld() == 12)` wird eventuell nie erfüllt, da es vorkommen kann, dass die Bewegungs-ID in einem Schritt von 11.998 auf 12.013 erhöht wird und niemals exakt den erwarteten Wert (12) einnimmt. Sie sollten stattdessen `wait(getMoveld() >= 12)` schreiben.

Beispiel

Dieses Beispiel zeigt, wie sich die Bewegungs-ID auf einem einfachen Pfad ändert:

```
nIdA = movel(pA, tTool, mDesc)
nIdB = movel(pB, tTool, mDesc)
waitEndMove()
nId = getMoveId()
```

Während der Ausführung dieses Programms:

Nehmen Sie an, dass der gelieferte Wert nldA 15 beträgt. Dann beträgt nldB 16: die Bewegungs-ID wird mit jeder Bewegungsanweisung automatisch um eins erhöht.

- wenn `getMoveld()` 15.8 entspricht, ist die Roboterposition auf 80 % der Bewegung zu Punkt pA.
- wenn `getMoveld()` 16.572 entspricht, ist die Roboterposition auf 57.2 % der Bewegung zu Punkt pB.
- wenn `getMoveld()` 17 entspricht, ist die Roboterposition auf 100 % der Bewegung 16, also auf Punkt pB.

Der Wert von nld nach `waitEndMove()` beträgt deshalb nldB+1=17.

Siehe auch

`num movej(joint jPosition, tool tTool, mdesc mDesc)`
`num movel(point pPosition, tool tTool, mdesc mDesc)`
`num movec(point plIntermediate, point pTarget, tool tTool, mdesc mDesc)`

num setMoveld(num nMoveld)

Funktion

Diese Anweisung ändert die Bewegungs-ID für die nächste Bewegungsanweisung. Es ist sinnvoll so vorzugehen, dass die gleiche Bahn immer die gleichen Werte für die Bewegungs-IDs verwendet. Nach `resetMotion` wird die Bewegungs-ID automatisch auf 0 zurückgesetzt.

Nach Verwendung von `setMoveld()` oder `resetMotion()` kann die Beziehung zwischen einer Bewegungs-ID und einer Bewegungsanweisung unsicher werden: mehrere aufgezeichnete Bewegungen können dann dieselbe Bewegungs-ID aufweisen. Deshalb sollte `setMoveld()` keinen Wert erhalten, der auch die Bewegungs-ID eines auf Ausführung wartenden Bewegungsbefehl ist.

Beispiel

```
resetMotion()
nId1 = getMoveId()
setMoveId(1000)
nId2 = getMoveId()
nId3 = movel(pA, tTool, mDesc)
nId4 = movel(pB, tTool, mDesc)
waitEndMove()
nId5 = getMoveId()
```

Nach der Ausführung dieses Programm haben wir Folgendes:

- nld1 ist 0, da Bewegungs-ID nach `resetMotion()` auf 0 gesetzt wurde
- nld2 ist 0: Bewegungs-ID wurde gerade mit `setMoveld()` geändert
- nld3 ist 1000: eine Bewegungsanweisung liefert die zuvor definierte Bewegungs-ID und vergrößert sie für die nächste Bewegung
- nld4 ist 1001: die Bewegungs-ID wurde durch die vorherige Bewegungsanweisung vergrößert
- nld5 ist 1002: nach `waitForEndMove()` werden 100 % der Bewegung 1001 hier ausgeführt, die Bewegungs-ID lautet dann $1001+1 = 1002$

Siehe auch

`num getMoveld()`

`void resetMotion(), void resetMotion(joint jStartingPoint)`

KAPITEL 11

EINSTELLUNG DER ROBOTERNUTZLAST

Es ist möglich, die tatsächlichen Lasten, die vom Arm bewegt werden, festzulegen.

Dadurch wird die statische Präzision für die Arme mit der Option "absolute Kalibrierung" verbessert, da diese Option den Flexibilitätsausgleich aktiviert.

Im allgemeinen wird dadurch auch das Verhalten des Arms verbessert, wenn die Applikation sehr große Dynamik erfordert, da die Drehmomentvorsteuerungen bei angegebener Last genauer sind.

11.1. PRINZIP

Die Lasten werden mit einem Gewicht und einer Trägheit definiert. Für jedes Gelenk des Arms gibt es drei Parametersätze:

- Die Systemparameter. Sie werden in der Datei system.zfx definiert und beschreiben die Armlast selbst. Sie können vom Benutzer nicht gelesen oder geändert werden.
- Die Zellenparameter. Sie definieren die am Arm montierten statischen Lasten. Zum Beispiel stellen ein Greifer oder eine Ausrüstung, die am Vorderarm montiert ist, eine statische Last dar. Sie werden in cell.cfx gespeichert und während des Startvorgangs der Steuerung einmal initialisiert. Siehe **Benutzerhandbuch der Steuerung** zur Einstellung der Zellenlasten.

Wenn keine Nutzlast in cell.cfx festgelegt ist, wird eine **Standardnutzlast** beim Boot benutzt. In diesem Fall wird eine Warnmeldung im Protokoll angezeigt.

Diese **Standardnutzlast** wird gelöscht, sobald eine Nutzlast über die Funktion **setPayload()** festgelegt wird.

- Die "User"-Parameter repräsentieren die **Nutzlast** des Roboters: Das sind typischerweise die Teile, die während der Applikation aufgenommen und abgelegt werden. Die **Nutzlast** kann von der **VAL 3** Applikation dynamisch geändert werden. Sie werden beim Starten des Systems gelöscht.

Das dynamische Modell der Steuerung nutzt eine Kombination aus den oben genannten Parametern.

Diese Funktion wird nicht von allen Armen unterstützt. Dies kann entweder über die **VAL 3**-Applikation (siehe **VAL 3**-Funktionen unten) oder über das Protokoll überprüft werden, wenn die tatsächliche Nutzlast des Roboters beim Systemboot aufgezeichnet wird: Falls die Datei keine Informationen zur Nutzlast enthält, wird diese Funktion vom Arm nicht unterstützt.

11.2. ANWEISUNGEN

**num setPayload(tool tTool, num nMass, trsf trGravityCenter,
 num& nInertiaMatrix[])**

Funktion

Diese Funktion verändert die Nutzlast des Roboters.

| Parameter | Beschreibung |
|------------------------|---|
| tTool | Das Tool, in dem die Koordinaten des Schwerpunkts angegeben sind. |
| nMass | Die Nutzlast in kg. |
| trGravityCenter | Die Position des Schwerpunkts in tTool-Koordinaten. Nur die Felder x, y und z werden benutzt. |
| nInertiaMatrix | Die Trägheitsmatrix der Nutzlast an ihrem Schwerpunkt in Tool-Koordinationen (Einheit kg.m ²). Verschiedene Formate sind möglich: <ul style="list-style-type: none"> • 2-dims 3x3 Array, • 1-dim 9 Elemente-Array, • 1-dim 3 Elemente-Array (diagonale Matrix), • 1-dim 1 Elemente-Array (keine Trägheit). |

Diese Funktion gibt einen Fehlercode zurück:

| Code | Beschreibung |
|-------------|--|
| 0 | Änderung vorgenommen. |
| -1 | Die Nutzlast wird für diesen Arm nicht unterstützt. |
| -3 | Die Trägheit ist bereits geändert worden (der Vorgang des Umschaltens von einer Nutzlast zur anderen hat die Dauer der Boxcar-Zeit). |
| -4 | Die Eingangsmasse ist negativ. |
| -5 | Die Eingangsträgheitsmatrix ist asymmetrisch. |

**num getPayload(tool tTool, num& nMass, trsf& trGravityCenter,
num& nInertiaMatrix[]])**

Funktion

Diese Funktion dient zum Erhalt der tatsächlichen Nutzlast.

Die Parameter sind die gleichen wie setPayload().

Wenn nInertiaMatrix ein 1D-Array ist, wird es auf 9 skaliert und mit lxx, lxy, lxz, lyx, lyy, lyz, lzx, lzy und lzz gefüllt (außer wenn die Trägheit symmetrisch ist und die Array-Größe bereits 3 ist).

Wenn nInertiaMatrix ein 2D-Array ist, wird es auf 3x3 skaliert und natürlich mit [lxx lxy lxz; lyx lyy lyz; lzx lzy lzz] gefüllt [Vollbildschirm].

Da lokale Arrays nicht skaliert werden können, tritt ein Laufzeitfehler auf, wenn nInertiaMatrix eine lokale Variable mit einer falschen Größe ist.

Die möglichen Rückgabecodes sind:

| Code | Beschreibung |
|-------------|---|
| 0 | Kein Fehler. |
| -1 | Die Nutzlast wird für diesen Arm nicht unterstützt. |

KAPITEL 12

SYSTEMEREIGNISSE

Codes der Systemereignisse

Die an der Steuerung festgestellten Ereignisse werden in der /log/system.log-Datei aufgezeichnet. Diese Ereignisse sind durch eine einmalige Nummer identifiziert. Zu bereits vorhandenen Ereigniscodes siehe Dokumentation der Steuerung.

void **getIds**(num& nEvtId)

Funktion

Diese Funktion ermöglicht das Auffinden der 20 letzten im System festgestellten Ereignisse im Array **nEvtId**.

- num& **nEvtId** ist ein Verweis auf ein Array, in dem das System die letzten festgestellten Ereignisse aufzeichnet. Die Ereignisse werden chronologisch sortiert, das neueste Ereignis wird am Index 0 des Array gespeichert. Wenn die Größe des Array kleiner als 20 ist, werden nur die neueren Ereignisse je nach verfügbarem Speicherplatz aufgezeichnet.

Beispiel

Das nachstehende Beispiel zeigt in jeder Sekunde die letzten erhaltenen Ereigniscodes an, sobald neue Ereignisse auftreten.

```
begin
  // 
  l_nLastId=-1
  do
    sOutput= " "
    delay(1)
    getIds(nEvtId)
    if nEvtId[0] != l_nLastId
      l_nLastId=nEvtId[0]
      for l_nI=0 to size (nEvtId)-1
        sOutput=sOutput+" ; "+nEvtId[l_nI]
      endFor
    endif
    until false
  end
```

num **getEvents**(num& x_nEvtNbr, num& x_nId[], num& x_nType[], string& x_sInfo[])

Funktion

Diese Funktion liest die letzten in der Steuerung aufgetretenen Systemereignisse. Das System behält die letzten 20 aufgetretenen Ereignisse im Speicher. Jedes Ereignis wird vom System beginnend mit 1 durchnummeriert, die Nummer wird nach jedem neuen Ereignis inkrementiert.

Diese Funktion kopiert die letzten im Puffer abgelegten Ereignisse beginnend mit der in **x_nEvtNbr** abgelegten Ereignisnummer in die angegebenen Parameter-Arrays **x_nId**, **x_nType**, **x_sInfo**. Die Zahl N des zu kopierenden Ereignisses wird durch die Mindestgröße der Parameter-Arrays angegeben und ist auf die Größe des internen Ereignispuffers (20) beschränkt. In den Arrays werden die Ereignisse von den ältesten bis zu den neuesten sortiert.

Wenn **x_nEvtNbr** sich auf eine Ereignisnummer bezieht, die nicht mehr im internen Puffer existiert (kleiner als das älteste Ereignis), kopiert die Funktion die **N** ältesten verfügbaren Ereignisse.

Der Parameter **x_nEvtNbr** gibt die Nummer des ersten, tatsächlich gelesenen Ereignisses an und die Funktion gibt die Nummer des letzten tatsächlich gelesenen Ereignisses zurück.

Wenn **x_nEvtNbr** sich auf eine Ereignisnummer bezieht, die nicht mehr im internen Puffer existiert (größer als das neueste Ereignis), kopiert die Funktion nichts. Der Parameter **x_nEvtNbr** bleibt unverändert, es wird keine Kopie gemacht und die Funktion gibt die Nummer des neusten verfügbaren Ereignisses zurück.

Eingangs-/Ausgangsparameter

x_nEvtNbr:

- Eingangseite: Nummer des ersten zu lesenden Ereignisses.
- Ausgang: Nummer des ersten tatsächlich gelesenen Ereignisses.

Ausgangsparameter

- **x_nId**: Array der Ereignis-ID.
- **x_nType**: Array des Ereignistyps (0=Popup, 1=Historie, 2=Protokoll).
- **x_sInfo**: Array der Ereignistextbeschreibung.

Rücksprung

Nummer des letzten kopierten Ereignisses oder Nummer des neuesten verfügbaren Ereignisses (siehe oben).

VAL 3 Systemereignis-Pooling-Beispiel

```
eventPolling()
begin
    // First loop read from event number 1
    l_nFirstEvt=-1
    do
        // Get pending events
        l_nEvt=l_nFirstEvt
        l_nLastEvt=getEvents(l_nFirstEvt, l_nId, l_nType, l_sInfo)
        // Compute number of read events
        l_nNbRead=l_nLastEvt-l_nFirstEvt+1
        // IF some events have been read
        if (l_nNbRead>0)
            // IF some events have been lost
            if (l_nEvt!=l_nFirstEvt)
                call proceedLost(l_nFirstEvt-l_nEvt, l_nEvt)
            endif
            call proceedEvent(l_nNbRead, l_nId, l_nType, l_sInfo, l_nFirstEvt)
            // Set event number for next loop
            l_nFirstEvt=l_nLastEvt+1
            // IF rollover
            elseIf (l_nNbRead!=0)
                l_nFirstEvt=1
            endif
            delay(0)
        until bStopLoop==true
    end
```

KAPITEL 13

OPTIONEN

13.1. ALTER: BEWEGUNGSSTEUERUNG IN ECHTZEIT

ACHTUNG:

Die Alter-Funktion erfordert hohe Echtzeitressourcen. Je nach CPU-Typ kann es sein, dass die Systemzykluszeit reduziert wird.

Translationsbewegung Alter

13.1.1. PRINZIP

Durch die kartesische Änderung einer Bahn kann eine unmittelbar effektive Koordinatentransformation (Translation, Rotation, Rotation am Werkzeugnullpunkt) auf den Weg ausgeübt werden. Diese Funktion ermöglicht die Änderung der theoretischen Bahn mithilfe eines externen Fühlers, um z.B. die genaue Mitverfolgung der Form eines Teils zu gewährleisten oder an einem in Bewegung befindlichen Teil einzugreifen.

13.1.2. PROGRAMMIERUNG

Bei der Programmierung wird zuerst die theoretische Bahn festgelegt und anschließend die entsprechende Abweichung in Echtzeit.

Die theoretische Bahn wird, genauso wie die standardmäßigen Bewegungen, mithilfe der Anweisungen `alterMove()`, `alterMovej()` und `alterMovec()` programmiert. Mehrere veränderbare Bewegungen können aufeinander folgen, oder bestimmte veränderbare Bewegungen mit nicht veränderbaren abwechseln. Wir werden eine veränderbare Bahn als Abfolge veränderbarer Bewegungsbefehle zwischen zwei nicht veränderbaren Bewegungsbefehlen definieren.

Die eigentliche Änderung wird mit der Anweisung `alter()` programmiert. Je nach durchzuführender Koordinatentransformation sind unterschiedliche alter-Modi möglich, der Modus wird mit der Anweisung `alterBegin()` definiert. Schließlich dient die Anweisung `alterEnd()` auch dazu anzugeben, wie die Änderung beendet werden soll, entweder vor der Beendigung der theoretischen Bewegung, um die folgende nicht veränderbare Bewegung unterbrechungsfrei sequentiell ordnen zu können, oder danach, um den Arm mit alter bewegen zu können, während die theoretische Bewegung unterbrochen ist.

Die anderen Anweisungen zu Bewegungsbefehlen bleiben im Modus alter gültig.

ACHTUNG:

Die Anweisungen `waitEndMove`, `open` und `close` warten das Ende der theoretischen und nicht der veränderten Bewegung ab. Die Ausführung von **VAL 3** kann dann nach `waitEndMove` wieder aufgenommen werden, auch wenn der Arm aufgrund einer Alter-Funktion noch in Bewegung ist.

13.1.3. RANDBEDINGUNGEN

Synchronisierung, Desynchronisierung: Da der Befehl alter sofort ausgeführt wird, muss das Wechseln der Änderung so gesteuert werden, dass die sich ergebende Bahn des Arms keinerlei Unterbrechung oder Geräusche aufweist:

- Erhebliche Wechsel der Änderung können nur schrittweise mit einem spezifischen Annäherungsbefehl vorgenommen werden.
- Zur Beendigung der Änderung muss die Änderungsgeschwindigkeit gleich null sein. Dies wird schrittweise durch einen spezifischen Anhaltebefehl erreicht.

Synchronbefehl: Der Controller sendet den Verstärkern alle 4 ms Befehle zu Position und Geschwindigkeit. Folglich muss der Befehl alter so mit dieser Kommunikationsperiode synchronisiert werden, dass die Änderungsgeschwindigkeit unter Kontrolle bleibt. Dazu wird eine synchrone **VAL 3**-Task verwendet (siehe Kapitel Tasks). Gleichermaßen muss manchmal am Fühlereingang gefiltert werden, wenn die Daten zu starke Geräusche aufweisen oder ihre Probenperiode nicht mit der Periode des Controllers synchronisiert ist.

Progressives sequentielles Ordnen: Die erste nicht veränderbare Bewegung auf einer veränderbaren Bahn kann erst nach Ausführung von alterEnd berechnet werden. Folglich besteht das Risiko, dass der Arm sich bei zu rascher Ausführung von alterEnd nach Beendigung der veränderbaren Bewegung verlangsamt oder sogar in der Nähe dieses Punkts anhält, während die folgende Bewegung noch nicht berechnet wurde.

Außerdem bringt die Fähigkeit, die folgende Bewegung im Voraus zu berechnen Beschränkungen der veränderten Bahn nach Ausführung von alterEnd mit sich: Sie muss die gleiche Konfiguration behalten, und man muss sich versichern, dass alle Gelenkpunkte in derselben Achsdrehung bleiben. Es ist also möglich, dass während der Bewegung ein Fehler auftritt, was nicht möglich gewesen wäre, wenn alterEnd nicht im Voraus ausgeführt worden wäre.

13.1.4. SICHERHEIT

Die Änderung des Benutzers kann jederzeit ungültig werden: Ziel außer Reichweite, zu hohe Geschwindigkeit oder Beschleunigung. Wenn das System Situationen dieser Art feststellt, wird eine Fehlermeldung erstellt und der Arm plötzlich in der letzten gültigen Position immobilisiert. Vor dem Neustarten der Funktion muss die Bewegung wieder hergestellt werden.

Bei Deaktivierung der Armbewegung während einer Bewegung (Halte-Modus, Stopp- oder Nothaltanfrage) wird der Anhaltebefehl auf die theoretischen und die standardmäßigen Bewegungen ausgeübt. Nach einer gewissen Wartezeit wird auch der Modus alter automatisch deaktiviert, um das komplette Anhalten des Arms zu garantieren. Wenn der Anhaltezustand verschwindet, setzt die Bewegung wieder ein und der Modus alter wird automatisch reaktiviert.

13.1.5. BEGRENZUNGEN

Null-Bewegung: Bewegungen gleich null (Bewegungsziel befindet sich in Startposition) werden vom System nicht berücksichtigt. Folglich ist eine Bewegung, die nicht Null ist, erforderlich, um den Alter-Modus aufzurufen. Eine Bewegungsdistanz von 0.001 mm ist dazu genug.

Roboter-Konfiguration: Die für die geänderte Bahn anzuwendende Konfiguration kann nicht spezifiziert werden; das System verwendet nämlich stets dieselbe Konfiguration. Die Konfiguration des Arms innerhalb einer geänderten Bahn kann also nicht geändert werden (auch mit der Anweisung alterMovej).

Veränderbare Pfad-Sequenzierung: Es ist nicht möglich, die Änderung mit alterBegin() nach alterEnd() ohne eine Standardbewegung (moveJ, moveC, moveL) oder ein eingefügtes resetMotion() neu zu starten. Jeder kontinuierlich veränderbare Pfad unterstützt nur einen alterBegin().

13.1.6. ANWEISUNGEN

num alterMovej(joint jPosition, tool tTool, mdesc mDesc)

num alterMovej(point pPosition, tool tTool, mdesc mDesc)

Funktion

Diese Anweisung übergibt einen veränderbaren Bewegungsbefehl für die Gelenkposition (eine Linie im Bereich der Gelenkposition). Sie liefert die dieser Bewegung zugewiesene Bewegungs-ID zurück und erhöht die Bewegungs-ID für den nächsten Bewegungsbefehl um eins.

Parameter

| | |
|----------------------------|---|
| jPosition/pPosition | Ausdruck vom Typ Point oder Joint zur Festlegung der Position am Ende der Bewegung. |
| tTool | Ausdruck vom Typ Werkzeug zur Festlegung des während der Bewegung zum kartesischen Geschwindigkeitsbefehl verwendeten Werkzeugnullpunkts. |
| mDesc | Ausdruck mDesc zur Festlegung von Geschwindigkeitssteuerung und Blending-Parameter der Bewegung. |

Details

Diese Anweisung verhält sich genauso wie die Anweisung movej, bis auf dass sie nicht den Modus alter für die Bewegung erlaubt. Weitere Informationen siehe movej.

num alterMoveL(point pPosition, tool tTool, mdesc mDesc)

Funktion

Diese Anweisung übergibt einen linearen veränderbaren Bewegungsbefehl (eine Linie im kartesischen Raum). Sie liefert die dieser Bewegung zugewiesene Bewegungs-ID zurück und erhöht die Bewegungs-ID für den nächsten Bewegungsbefehl um eins.

Parameter

| | |
|------------------|--|
| pPosition | Ausdruck vom Typ Point zur Festlegung der Position am Ende der Bewegung. |
| tTool | Ausdruck vom Typ Werkzeug zur Festlegung des während der Bewegung zum kartesischen Geschwindigkeitsbefehl verwendeten Werkzeugnullpunkts. Am Ende der Bewegung befindet sich der Werkzeugnullpunkt in der spezifizierten Zielposition. |
| mDesc | Ausdruck mdesc zur Festlegung von Geschwindigkeitssteuerung und Blending-Parameter der Bewegung. |

Details

Diese Anweisung verhält sich genauso wie die Anweisung moveL, bis auf dass sie nicht den Modus alter für die Bewegung erlaubt. Weitere Informationen siehe moveL.

num alterMoveC(point plIntermediate, point pTarget, tool tTool, mdesc mDesc)

Funktion

Diese Anweisung übergibt einen veränderbaren Kreisbewegungsbefehl. Sie liefert die dieser Bewegung zugewiesene Bewegungs-ID zurück und erhöht die Bewegungs-ID für den nächsten Bewegungsbefehl um eins.

Parameter

| | |
|-----------------------|--|
| plIntermediate | Ausdruck vom Typ Point zur Festlegung eines mittleren Kreispunkts |
| pTarget | Ausdruck vom Typ Point zur Festlegung der Position am Ende der Bewegung. |
| tTool | Ausdruck vom Typ Werkzeug zur Festlegung des während der Bewegung zum kartesischen Geschwindigkeitsbefehl verwendeten Werkzeugnullpunkts. Am Ende der Bewegung befindet sich der Werkzeugnullpunkt in der spezifizierten Zielposition. |
| mDesc | Ausdruck mdesc zur Festlegung von Geschwindigkeitssteuerung und Blending-Parameter der Bewegung. |

Details

Diese Anweisung verhält sich genauso wie die Anweisung moveC, bis auf dass sie nicht den Modus alter für die Bewegung erlaubt. Weitere Informationen siehe moveC.

num alterBegin(frame fAlterReference, mdesc mMaxVelocity)**num alterBegin(tool tAlterReference, mdesc mMaxVelocity)****Funktion**

Diese Anweisung initialisiert den alter-Modus für die auszuführende Bahn.

Parameter

fAlterReference/tAlterReference Ausdruck vom Typ Rahmen oder Werkzeug zur Festlegung der Referenz des Abstands alter.

mMaxVelocity Ausdruck mdesc zur Festlegung der Parameter für die Überprüfung von Sicherheit und Abstand alter.

Details

Der mit alterBegin gestartete Modus alter kann nur mit einem alterEnd-Befehl oder mit resetMotion beendet werden. Beim Erreichen des Endes einer veränderbaren Bahn bleibt der Modus alter bis zur Ausführung von alterEnd aktiv.

Der Ausdruck trsf des Befehls alter dient zur Festlegung einer Transformation der gesamten Bahn um alterReference:

- Die gesamte Bahn dreht sich um den Mittelpunkt des Rahmens oder den Werkzeugnullpunkt unter Verwendung des Rotationsteils von trsf.
- Die Bahn wird anschließend durch das Translationsteil von trsf einer Translation unterzogen.

Die trsf-Koordinaten des Befehls alter werden in der Datenbank alterReference definiert.

Bei Verwendung eines Rahmens als Referenz, wird alterReference im Raum (World) festgelegt. Dieser Modus muss benutzt werden, wenn der Abstand der Bahn bekannt ist oder im kartesischen Raum gemessen wird (mobiles Teil wie z.B. Spur des Transportbands).

Bei Verwendung eines Werkzeugs als Referenz, ist alterReference bezüglich des Werkzeugnullpunkts fest. Dieser Modus muss benutzt werden, wenn der Abstand der Bahn bekannt ist oder bezüglich des Werkzeugnullpunkts gemessen wird (z.B. am Werkzeug angebrachter Fühler für die Teileform).

Der Bewegungsdeskriptor wird zur Bestimmung der Joint-Geschwindigkeit und der maximalen kartesischen Geschwindigkeit auf der veränderten Bahn benutzt (mithilfe der Felder vel, tvel und rvel des Bewegungsdeskriptors). Übersteigt die veränderte Geschwindigkeit die spezifizierten Grenzen, so wird eine Fehlermeldung erstellt und der Arm auf der Bahn gestoppt.

Die Felder accel und decel des Bewegungsdeskriptors steuern die Anhaltedauer wenn die Bewegung gestoppt wird (eStop, Haltemodus, VAL 3 [stopMove\(\)](#)): Die Änderung der Bahn muss durch Verwendung der folgenden Abbremsparameter gestoppt werden (siehe alterStopTime).

alterBegin überträgt einen numerischen Wert, der das Ergebnis der Anweisung angibt:

| | |
|-----------|--|
| 1 | alterBegin wurde erfolgreich ausgeführt |
| 0 | alterBegin wartet auf den Beginn der veränderbaren Bewegung |
| -1 | alterBegin wurde nicht berücksichtigt, weil der Modus alter schon gestartet wurde |
| -2 | alterBegin wurde nicht akzeptiert (die Option alter ist nicht aktiviert) |
| -3 | alterBegin wurde nicht akzeptiert, weil die Bewegung fehlerhaft ist. Ein resetMotion ist erforderlich. |

Siehe auch

[num alterEnd\(\)](#)

[num alter\(trsf trAlteration\)](#)

[num alterStopTime\(\)](#)

num alterEnd()

Funktion

Diese Anweisung beendet den alter-Modus und bewirkt, dass die aktuelle Bewegung unveränderbar ist.

Details

Wenn alterEnd nach Erreichen des Endes der veränderbaren Bahn ausgeführt wird, startet die nächste nicht veränderbare Bewegung (falls vorhanden) sofort.

Wenn alterEnd vor Erreichen des Endes der veränderbaren Bewegung ausgeführt wird, gilt der aktuelle Wert des Abstands alter für den Rest der veränderbaren Bahn bis zur nächsten nicht veränderbaren Bewegung. Es ist nicht möglich, auf derselben veränderbaren Bahn in den Modus alter zurückzukehren.

Die nächste nicht veränderbare Bewegung (falls vorhanden) wird ab der Ausführung von alterEnd berechnet, um den Übergang zwischen veränderbarer Bahn und nicht veränderbarer Bewegung unterbrechungslos zu vollziehen.

alterEnd überträgt einen numerischen Wert, der das Ergebnis der Anweisung angibt:

| | |
|-----------|--|
| 1 | alterEnd wurde erfolgreich ausgeführt |
| -1 | alterEnd wurde nicht berücksichtigt, weil der Modus alter noch nicht gestartet wurde |
| -3 | alterEnd wurde nicht akzeptiert, weil die Bewegung fehlerhaft ist. Ein resetMotion ist erforderlich. |

Siehe auch

[num alterBegin\(frame fAlterReference, mdesc mMaxVelocity\)](#)

[num alterBegin\(tool tAlterReference, mdesc mMaxVelocity\)](#)

num alter(trsf trAlteration)

Funktion

Diese Anweisung übergibt eine neue Änderung der veränderbaren Bahn.

Details

Die durch die trsf-Änderung bewirkte Transformation ist von dem durch die Anweisung alterBegin ausgewählten Modus alter abhängig. Die Koordinaten der Änderung werden in dem durch die Anweisung alterBegin spezifizierten Rahmen oder Werkzeug definiert.

Die Veränderung wird vom System alle 4 ms ausgeführt: Bei Ausführung mehrerer alter-Anweisungen in weniger als 4 ms gilt die zuletzt ausgeführte. Meistens muss die alter-Anweisung in einer synchronen Task ausgeführt werden, um eine Aktualisierung der Änderung alle 4 ms zu bewirken.

Die Änderung muss sehr sorgfältig berechnet werden, damit die sich daraus ergebenden Befehle zu Position und Geschwindigkeit des Arms ununterbrochen und geräuschlos ausgeführt werden. Manchmal muss zuvor der Fühlereingang gefiltert werden, um gewünschte Qualität auf der Bahn und Verhalten des Arms zu erreichen.

Wenn die Bewegung gestoppt ist (Halte-Modus, Notaus, Anweisung [stopMove\(\)](#)), wird die Änderung der Bahn so lange verriegelt, bis sämtliche Anhaltebedingungen beseitigt wurden.

Wenn die Änderung der Bahn ungültig ist (nicht erreichbare Position, Geschwindigkeitsgrenzen überschritten) hält der Arm plötzlich auf der letzten gültigen Position an und der Modus alter wird in Fehlerposition verriegelt. Vor dem Neustarten des Betriebs ist ein resetMotion erforderlich. Die Geschwindigkeitsgrenzen der Bewegung alter werden in der Anweisung alterBegin festgelegt.

Alter überträgt einen numerischen Wert, der das Ergebnis der Anweisung angibt:

| | |
|-----------|---|
| 1 | alter wurde erfolgreich ausgeführt. |
| 0 | alter wartet auf den Neustart der Bewegung (alterStopTime ist gleich null). |
| -1 | alter wurde nicht berücksichtigt, weil der Modus alter noch nicht gestartet oder schon beendet wurde. |
| -2 | alter wurde nicht akzeptiert (die Option alter ist nicht aktiviert). |
| -3 | alter wurde nicht akzeptiert, weil die Bewegung fehlerhaft ist. Ein resetMotion ist erforderlich. |

Siehe auch

num alterBegin(frame fAlterReference, mdesc mMaxVelocity)
num alterBegin(tool tAlterReference, mdesc mMaxVelocity)
void taskCreateSync string sName, num nPeriod, bool& bOverrun, program(...)

num alterStopTime()

Funktion

Diese Anweisung liefert die verbleibende Zeit bei Anhaltevorgängen, in der noch alter-Vorgänge akzeptiert werden.

Details

Findet ein Anhaltevorgang statt, so nimmt das System eine Abschätzung der zum Stoppen des Arms notwendigen Zeit vor, wenn die von alterBegin spezifizierten Parameter accel und decel des Bewegungsdeskriptors verwendet werden. Die notwendige Mindestdauer für dieses Anhalten und die vom System vorgeschriebene Zeit (normalerweise 0.5s bei einem eStop) werden von alterStopTime übertragen.

Überträgt alterStopTime einen negativen Wert, gibt es keine drohende Anhaltebedingung. Ist der von alterStopTime übertragene Wert ungültig, wird der Befehl alter so lange verriegelt, bis sämtliche Anhaltebedingungen neu initialisiert worden sind.

Ist der Modus alter nicht aktiviert, überträgt alterStopTime einen ungültigen Wert.

Siehe auch

num alterBegin(frame fAlterReference, mdesc mMaxVelocity)
num alterBegin(tool tAlterReference, mdesc mMaxVelocity)
num alter(trsf trAlteration)

13.2. OEM-LIZENZKONTROLLE

13.2.1. GRUNDLAGEN

Eine OEM-Lizenz ist ein controller-spezifischer Schlüssel um die Verwendung eines **VAL 3**-Projekts auf einige ausgewählte Roboter-Controller zu beschränken.

Mit **Stäubli Robotics Suite**(*) wird ein Tool geliefert, um ein geheimes OEM-Passwort in eine öffentliche, controller-spezifische OEM-Lizenz zu codieren, die als Software-Option auf dem Controller installiert werden kann. Durch Verwendung der Anweisung **getLicence()** kann ein Projekt oder eine Bibliothek testen, ob die OEM-Lizenz installiert ist und somit sicherstellen, dass sie nur von den ausgewählten Roboter-Controllern verwendet wird.

Um das OEM-Passwort geheim zu halten und den Code zum Testen der Lizenz zu schützen, muss die Anweisung **getLicence()** in einer verschlüsselten Bibliothek verwendet werden.

Der Vorführungsmodus von OEM-Lizenzen wird unterstützt; in diesem Fall wird der Controller einfach mit dem "Demo"-Schlüssel konfiguriert und die Anweisung **getLicence()** meldet dies zurück. Mit dem **VAL 3** Emulator, reicht der "Demo"-Modus aus, um die OEM Lizenz vollständig zu aktivieren.

Die Anweisung **getLicence()** ist eine **VAL 3**-Option und erfordert die Installation einer Laufzeitlizenz auf dem Controller. Wenn diese Laufzeitlizenz nicht definiert ist, meldet **getLicence()** einen Fehlercode.

(*) Dieses Tool, die ausführbare Datei `encrypttools.exe`, erfordert zur Verwendung eine spezifische **Stäubli Robotics Suite**-Lizenz.

13.2.2. ANWEISUNGEN

string getLicence(string sOemLicenceName, string sOemPassword)

Funktion

Diese Anweisung gibt den Status der angegebenen OEM-Lizenz zurück:

| | |
|-----------------------------|--|
| "oemLicenceDisabled" | Die VAL 3 -Laufzeitlizenz "oemLicence" ist auf dem Controller nicht aktiviert: die OEM-Lizenz kann nicht getestet werden. |
| "" | Die OEM-Lizenz sOemLicenceName ist nicht aktiviert (nicht definiert oder ungültiges Passwort). |
| "demo" | Die OEM-Lizenz sOemLicenceName ist im Demomodus aktiviert. |
| "enabled" | Die OEM-Lizenz sOemLicenceName ist aktiviert. |

Siehe auch

Verschlüsselung

13.3. ABSOLUTER ROBOTER

13.3.1. PRINZIP

Ein 'absoluter Roboter' ist ein Roboter, der eine armspezifische Identifikation der geometrischen Parameter nutzt (oft als 'DH-Parameter bekannt'). Diese Parameter sind für jeden Roboter spezifisch und entsprechen der tatsächlichen Ausrichtung und den Abmessungen jedes Gelenks. Ein absoluter Roboter besitzt eine höhere Genauigkeit, um kartesische Positionen zu erreichen, die durch ein CAD-Werkzeug angegeben oder in VAL 3 berechnet wurden (wie z.B. Positionen in einer Palette). Kartesische Kurven (lange Linien, Kreise) werden ebenfalls genauer. Absolute Kalibrierung ändert nichts an der Wiederholgenauigkeit des Arms.

Die DH-Parameter bestehen aus einem Satz Translationen (a, b und d entlang der Achsen X, Y, Z,) und Rotationen (Alpha, Beta, Theta um die Achsen X, Y und Z)

Die Sequenz dieser Translationen und Rotationen wird so definiert dass die Joint-Position {j1, j2, j3, j4, j5, j6} der kartesischen Position pCart am Flansch-Mittelpunkt folgendermaßen entspricht:

```
pCart.trsf = {0,0,0,0,0, j1+theta[0]}
* {a[0], b[0], d[0], alpha[0], beta[0], j2+theta[1]}
* {a[1], b[1], d[1], alpha[1], beta[1], j3+theta[2]}
* {a[2], b[2], d[2], alpha[2], beta[2], j4+theta[3]}
* {a[3], b[3], d[3], alpha[3], beta[3], j5+theta[4]}
* {a[4], b[4], d[4], alpha[4], beta[4], j6+theta[5]}
* {a[5], b[5], d[5], alpha[5], beta[5], 0}
* {0, 0, d[6], 0, 0, 0}
```

Der d[6]-Parameter ist nur für spezielle Roboter-Handgelenke erforderlich.

13.3.2. BETRIEB

Die geometrischen Parameter müssen mithilfe eines separaten Messwerkzeugs identifiziert werden (z.B. ein Lasertracker). Die VAL 3-Sprache bietet keine Werkzeuge zur Unterstützung dieses Messvorgangs, aber die Möglichkeit die gemessenen geometrischen Parameter mit sofortiger Wirkung auf den Roboter anzuwenden. Die Parameter werden ebenfalls in der Armkonfigurationsdatei gespeichert, sodass sie beim nächsten Neustart automatisch wiederhergestellt werden.

Die geometrischen Parameter können in einer laufenden VAL 3-Applikation geändert werden. Dadurch kann die Geometrie des Arms während des Produktionszyklus angepasst werden, wenn die Roboterzelle ein entsprechendes Messwerkzeug enthält.

13.3.3. BEGRENZUNGEN

Die geometrischen Parameter in einer laufenden VAL 3-Applikation können nur geändert werden, wenn der Bewegungsgenerator leer ist (keine anstehende Bewegung).

Die veränderte Geometrie eines absoluten Roboters hat komplexere mathematische Eigenschaften als eine standardmäßige Geometrie. Die Definition der Armkonfiguration (Konfigurationstyp) kann nicht mehr exakt erfolgen. Die Konvertierung der kartesischen Position in eine entsprechende Joint-Position kann in der Nähe der Achs-Grenzen oder an Einzelpositionen fehlschlagen.

13.3.4. ANWEISUNGEN

void getDH (num& theta[], num& d[], num& a[], num& alpha[], num& beta[])

void getDefaultDH(num& theta[], num& d[], num& a[], num& alpha[], num& beta[])

Funktion

Diese Anweisungen liefern die DH-Parameter des Arms in den angegebenen Arrays zurück. Die Parameter d und a sind Translationen in mm; die Parameter Theta, Alpha und Beta sind Winkelangaben in Grad. [getDH\(\)](#) liefert die aktuellen DH-Parameter des mit dem Controller verbundenen Arms. [getDefaultDH\(\)](#) liefert die standardmäßigen DH-Parameter für den Armtyp.

Siehe auch

[bool setDH\(num& theta\[\], num& d\[\], num& a\[\], num& b\[\], num& alpha\[\], num& beta\[\]\)](#)

bool setDH(num& theta[], num& d[], num& a[], num& b[], num& alpha[], num& beta[])

Funktion

Diese Anweisung wird nur mit einer speziellen Laufzeitlizenz aktiviert. Sie ändert die Armgeometrie mit DH-Parametern. Die Parameter d, a und b sind Translationen in mm; die Parameter Theta, Alpha und Beta sind Winkelangaben in Grad. Die Änderungen erfolgen sofort und werden auch in der Armkonfigurationsdatei gespeichert, damit die veränderte Geometrie auch beim nächsten Neustart effektiv ist.

Die Anweisung liefert 'true', falls die Änderung erfolgreich angewendet wird bzw. 'false', wenn die neue Geometrie nicht angewendet wird, weil sie zu sehr von den standardmäßigen Geometrieparametern abweichen. [setDH\(\)](#) wartet bis keine Bewegungsanweisungen vorliegen, um diesen Vorgang auszuführen.

Die Größe der DH-Arrays muss der Anzahl an Roboterachsen entsprechen. Ein zusätzlicher Eintrag im Array d kann erforderlich sein, um die Flanschabmessungen zu ändern: fehlt er, meldet [setDH\(\)](#) 'false' und eine Diagnosemeldung wird zum Ereignisprotokoll gesendet.

Siehe auch

[void getDH \(num& theta\[\], num& d\[\], num& a\[\], num& alpha\[\], num& beta\[\]\)](#)
[void getDefaultDH\(num& theta\[\], num& d\[\], num& a\[\], num& alpha\[\], num& beta\[\]\)](#)

13.4. KONTINUIERLICHE ACHSE

13.4.1. PRINZIP

Die Achse 6 (für RX/TX-Arme) oder Achse 4 (für RS/TS-Arme) kann 'kontinuierlich' in einer Roboter-Anwendung sein, wenn nur ihre Position innerhalb einer Umdrehung von Bedeutung ist: die Anzahl während des Zyklus ausgeführter Umdrehungen spielt keine Rolle. Dies gilt z.B. für Applikationen bei denen der Roboter ein Teil hält, das mit einem feststehenden Werkzeug bearbeitet wird.

Die continuousAxis-Option ermöglicht es die Anzahl der im vorherigen Zyklus ausgeführten Umdrehungen automatisch zurückzusetzen, bevor ein neuer Zyklus gestartet wird. Startet die Achse den Applikationszyklus beispielsweise in Position 0° und endet auf Position 720° (2 Drehungen) kann der nächste Zyklus die Position sofort auf 0° zurücksetzen und einen neuen Zyklus starten, ohne dass die Achse von 720° zurück auf 0° bewegt werden muss.

13.4.2. ANWEISUNGEN

joint resetTurn(joint jReference)

Funktion

Die Anweisung verhält sich wie die standardmäßige [resetMotion\(\)](#)-Anweisung, wartet bis der Arm gestoppt wird und passt die Position der kontinuierlichen Achse an, damit sie so nahe wie möglich an die übergebene Referenzposition herankommt. Sie liefert die tatsächliche Armposition nach Ausführung der Anweisung. Der Anpassungsvorgang erfolgt bei ein- oder ausgeschalteter Stromversorgung des Arms und dauert circa 50 ms. Die Anweisung [resetTurn\(\)](#) ändert die Kalibrierungsdaten des Arms. Beim nächsten Neustart des Controllers wird die Zero-Position der Achse automatisch reinitialisiert (Aktualisierung der armspezifischen Daten im Arm und auf der Festplatte des Controllers).

Siehe auch

[void resetMotion\(\)](#), [void resetMotion\(joint jStartingPoint\)](#)

KAPITEL 14

ANHANG

14.1. LAUFZEITFEHLERCODES

| Code | Beschreibung |
|------|--|
| -1 | Es existiert in der aktuellen Bibliothek oder Applikation keine Task mit dem angegebenen Namen |
| 0 | Die Task wird ohne Laufzeitfehler unterbrochen (Anweisung <code>taskSuspend()</code> oder Debug-Modus) |
| 1 | Die angegebene Task läuft |
| 10 | Ungültige numerische Berechnung (Division durch Null). |
| 11 | Ungültige numerische Berechnung (zum Beispiel $\ln(-1)$) |
| 20 | Zugriff auf ein Array mit höherem Index als die Arraygröße. |
| 21 | Zugriff auf ein Array mit negativem Index. |
| 23 | Verwendet eine Sammlung mit einem ungültigen Schlüssel. Ein Sammlungsschlüssel muss ein String-Wert sein, der nicht Null ist. |
| 25 | Zugriff auf eine Sammlung mit einem ungültigen Schlüssel. |
| 29 | Ungültiger Name der Task. Siehe Anweisung <code>taskCreate()</code> . |
| 30 | Der festgelegte Name entspricht keiner Task von VAL 3 . |
| 31 | Es existiert bereits eine Task gleichen Namens. Siehe Anweisung <code>taskCreate</code> . |
| 32 | Es werden nur 2 verschiedene Perioden für die synchronen Tasks unterstützt. Periode ändern. |
| 33 | Maximale Anzahl der VAL 3 synchronen Task-Zeilen erreicht. Die Anzahl der in einem Zyklus auszuführenden VAL 3 -Codezeilen ist auf 3000 für bVAL 3 synchrone Tasks begrenzt. |
| 34 | Die aktuelle Anweisung kann nicht gleichzeitig von 2 Tasks aufgerufen werden. |
| 40 | Der Speicherplatz genügt nicht für die Daten. |
| 41 | Der Ausführungsspeicher ist für die Task unzureichend. Siehe Größe des Ausführungsspeichers. |
| 45 | Watchdog wurde von <code>wdgRefresh()</code> aktualisiert, doch das System ist nicht für die Nutzung konfiguriert: <code>val3WatchdogPeriod <=0</code> |
| 60 | Maximale Ausführungszeit der Anweisung ist abgelaufen. |
| 61 | Interner Fehler des VAL 3 -Interpreters |
| 70 | Ungültiger Wert des Parameters der Anweisung. Siehe entsprechende Anweisung. |
| 71 | Ungültiger Vorgang an einer lokalen Variable. |
| 80 | Verwendung von Daten oder Programmen einer nicht im Speicher geladenen Bibliothek. |
| 81 | Die Kinematik ist nicht kompatibel: Verwendung eines point/joint/config, welcher nicht mit der Kinematik des Arms kompatibel ist. |
| 82 | Referenz-Frame oder -Tool einer Variablen gehören zu einer Bibliothek und sind nicht über den Geltungsbereich der Variablen zugänglich (Bibliothek nicht im Projekt der Variablen angegeben oder Referenzvariable ist privat). |
| 90 | Die Task kann an der festgelegten Stelle nicht wieder aufgenommen werden. Siehe Anweisung <code>taskResume()</code> . |
| 100 | Die im Bewegungsdeskriptor spezifizierte Geschwindigkeit ist ungültig (negativ oder zu groß). |
| 101 | Die im Bewegungsdeskriptor spezifizierte Beschleunigung ist ungültig (negativ oder zu groß). |
| 102 | Der im Motiondescriptor angegebene decel-Wert ist ungültig (negativ oder zu groß). |
| 103 | Der im Motiondescriptor angegebene tvel-Wert ist ungültig (negativ oder zu groß). |
| 104 | Der im Motiondescriptor angegebene rvel-Wert ist ungültig (negativ oder zu groß). |
| 105 | Der im Bewegungsdeskriptor spezifizierte Parameter reach ist ungültig (negativ). |
| 106 | Der im Bewegungsdeskriptor spezifizierte Parameter leave ist ungültig (negativ). |
| 122 | Schreibversuch auf einem Eingang des Systems. |
| 123 | Verwendung eines Ein-/Ausgangs dio, aio oder sio, der keinem Ein-/Ausgang des Systems zugeordnet ist. |
| 125 | Lese- oder Schreibfehler an einem dio, aio oder sio (Fehler auf Feldbus) |
| 126 | Addon: Ungültiges Eingangs-/Ausgangsformat (float/integer/signed/unsigned) Eine Variable des Typs aio VAL 3 wurde mit einem ungültigen Format benutzt. Zum Beispiel wird eine aio VAL 3 -Variable mit einem Float-Format in einer VAL 3 -Anweisung benutzt, während ein Ganzzahlformat erforderlich ist. |
| 150 | Diese Bewegungsanweisung lässt sich nicht ausführen: Eine vorher angeforderte Bewegung konnte nicht beendet werden (Punkt nicht erreichbar, singulärer Punkt, Konfigurationsproblem usw.) |
| 153 | Bewegungsbefehl nicht unterstützt |
| 154 | Ungültige Bewegungsanweisung: Bewegungsdeskriptor überprüfen. |
| 160 | Koordinaten des Tools flange nicht gültig |
| 161 | Koordinaten des Koordinatensystems world nicht gültig |
| 162 | Verwendung eines point ohne Referenzsystem. Siehe Definition. |

| Code | Beschreibung |
|-------------|---|
| 163 | Verwendung eines Koordinatensystems ohne Referenzsystem. Siehe Definition. |
| 164 | Verwendung eines Tools ohne Referenztool. Siehe Definition. |
| 165 | Referenzsystem oder Referenztool ungültig (globale Variable mit lokaler Variablen verbunden). |
| 170 | Addon: Kann einen mobilen Rahmen nicht mit einem anderen mobilen Rahmen verbinden. |
| 171 | Addon: Für diese Anweisung wird ein mobiler Rahmen erwartet. |
| 250 | Es ist keine Runtime-Lizenz für diese Anweisung installiert bzw. die Demo-Lizenz ist nicht mehr gültig. |
| 270 | Addon: Ungültiger Dateipfad. |
| 271 | Addon: Kann die Datei nicht lesen. |
| 290 | Addon: Ungültiger Bahntyp. |
| 301 | Addon: Externe Achsenposition erforderlich. Dieser Laufzeitfehler steht mit der Steuerfunktion der externen Achse in Verbindung. |
| 513 | Internaler Bewegungsgeneratorfehler. |
| 514 | Ungültiger Parameterwert. |
| 515 | Ungültiger Bewegungsgeneratorzustand. |
| 520 | Gelenkposition außerhalb der Softwaregrenzen. |
| 525 | Armposition untersagt: Die angegebene Gelenkposition hat zur Folge, dass sich einige Steuerpunkte innerhalb eines unzulässigen Bereichs befinden. |
| 530 | Keine Gelenklösung gefunden (keine Konvergenz). Kartesische Position wahrscheinlich außerhalb der Reichweite. |
| 531 | Kartesische Position außerhalb der Gelenkgrenzen. |
| 532 | Kartesische Position außerhalb des Arbeitsbereichs. |
| 533 | Kartesische Position mit festgelegter Konfiguration unerreichbar. |
| 534 | Ausrichtung mit diesem Arm unerreichbar. |
| 535 | Die gefundene Armposition befindet sich in einem unzulässigen Bereich: Die gefundenen Gelenklösungen zur Erreichung der angegebenen kartesischen Position hat zur Folge, dass sich einige Steuerpunkte innerhalb eines unzulässigen Bereichs befinden. |
| 536 | Die gefundenen Armpositionen befinden sich entweder außerhalb der Gelenkgrenzen oder innerhalb unzulässiger Bereiche: Es wurden mehrere Gelenklösungen zur Erreichung der angegebenen kartesischen Position gefunden (verschiedene Konfigurationen), doch keine befindet sich sowohl außerhalb der zulässigen Bereiche als auch innerhalb der Gelenkreichweite. |
| 540 | Kann die Einzigartigkeit in diesem Fall nicht kreuzen. |
| 541 | Internaler Fehler: Unterbrechung des kartesischen Pfads. |
| 542 | Unterbrechung der Konfiguration: Die erreichte Gelenkposition stimmt nicht mit der für die nächste Bewegung erforderliche Armposition überein. |
| 543 | Robot not at trajectory start position. |
| 544 | Der kartesische Pfad führt dazu, dass sich ein Gelenk mit mehreren Drehrichtungen außerhalb seines Bereichs bewegt. |
| 550 | Kreisbogen: Der Winkel zwischen den Start- und Zwischenpunkten überschreitet 180 Grad. |
| 551 | Kreisbogen: Der Winkel zwischen den Zwischen- und Endpunkten überschreitet 180 Grad. |
| 552 | Kreisbogen: Start-, Zwischen- und Endpunkte liegen zu eng beieinander. |
| 553 | Kreisbogen: Die Ausrichtungsänderung zwischen den Start- und Zwischenpunkten oder zwischen den Zwischen- und Endpunkten beträgt genau 180 Grad. Die Richtung der Bewegung ist nicht festgelegt. |
| 554 | Kartesisches Blending: Die Drehung ist zu weit. Die Bewegungsrichtung kann zweideutig sein. |
| 560 | Reserviert (Veraltete Funktion). |
| 561 | Addon: Kein Blending mit movejSync unterstützt. |
| 562 | Addon: 'joint' Blending nicht mit Verfolgungsbewegungen unterstützt. Stattdessen 'Cartesian' verwenden. |
| 563 | Addon: Kein Blending mit trajMove unterstützt. |
| 564 | Addon: Kein Blending mit splines unterstützt. |
| 565 | Addon: Ungültiges Blending in mdesc: 'Cartesian' Blending ist nicht möglich, wenn zwischen der Kinematik {externe Achsen} und {nur Arm} umgeschaltet wird. |
| 570 | Ungültige Daten für diesen Arm (falsche Anzahl Gelenke oder falsche Kinematik). |
| 571 | Ungültiger Kinematikkettenteil. |
| 572 | Addon: Teilmontagefehler: Ein resetMotion() ist erforderlich, um zum Bahnmodus mit getragenem Teil umzuschalten. |
| 573 | Addon: Teilmontagefehler: Ein resetMotion() ist erforderlich, um zum Bahnmodus mit getragenem Werkzeug umzuschalten. |
| 574 | Addon: Teilmontagefehler: Der Modus mit getragenem Teil wird für diese Bewegung nicht unterstützt. |
| 580 | Addon: Eine 'trackOn' Bewegung ist erforderlich, um eine Bandverfolgungssequenz zu starten. |

| Code | Beschreibung |
|-------------|--|
| 581 | Addon: Eine 'trackOff' Bewegung ist erforderlich, um eine Bandverfolgungssequenz zu beenden. |
| 582 | Addon: Der Bewegungsrahmen unterscheidet sich vom vorherigen. Eine 'trackon' Bewegung verwenden, um zwischen den Bewegungsrahmen umzuschalten. |
| 590 | Alter: Ungültiger Status, Befehl ignoriert. |
| 600 | Addon: Bewegungsgenerator gesperrt, um einen mobilen Rahmen freizugeben. |
| 610 | Reserviert (Veraltete Funktion). |
| 611 | Addon: Boxcar-Häufigkeit zu gering. |
| 999 | Protokoll prüfen, um eine Fehlerbeschreibung zu erhalten. |

ACHTUNG:

Laufzeitfehler-Code-Tags mit Addon sind nicht stabil und können in der zukünftigen Version geändert oder gelöscht werden.

INFORMATION:

Laufzeitfehlercodes zwischen 512 und 768 sind Bewegungsgeneratorfehler.

14.2. GRAFISCHE OBJEKTATTRIBUTE DER BENUTZERSEITE

Attributbeschreibung nachstehend im DOM-Vermerk (Eigenschaftsname in javascript https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Properties_Reference).

Die folgenden Kategorien sind festgelegt:

| Kategorie | Beschreibung |
|---------------------|---|
| forbidden | Darf nicht benutzt werden und kann zu einem unerwarteten Verhalten führen. |
| supported | Getestet und unterstützt. |
| unsupported | Nicht getestet, kann funktionieren oder auch nicht. |
| xor_attributes | Für eine Komponente, nur eines dieser Attribute darf nicht mehr verbunden werden. |
| writable_attributes | Attribute dieser Kategorie müssen mit der Richtung "w" verbunden werden. |
| def | Standardattribut des grafischen Objekts für das entsprechende Element |

Nachstehend eine Liste der Attribute der grafischen Objekte. Diese Liste ist mit der SRC-Version verknüpft, daher wird empfohlen, die Liste Ihrer SRC-Version in der nachstehenden Datei einzusehen:
sys/sp2/app/www-2.0/attributesConfig.json.

Das Element "*" im Element steht für alle Objekttypen.

```
{
  "elements": {
    "*": {
      "def": "innerHTML",
      "forbidden": [
        "id",
        "data-*",
        "hidden",
        "for",
        "name",
        "style.height",
        "style.left",
        "style.margin",
        "style.padding",
        "style.width",
        "style.top",
        "onabort",
        "onautocomplete",
        "onautocompleteerror",
        "onblur",
        "oncancel",
        "oncanplay",
        "oncanplaythrough",
        "onchange",
        "onclick",
        "onclose",
        "oncontextmenu",
        "oncuechange",
        "ondblclick",
        "ondrag",
        "ondragend",
        "ondragenter",
        "ondragexit",
        "ondragleave",
        "ondrop"
      ]
    }
  }
}
```

```
"ondragover",
"ondragstart",
"ondrop",
"ondurationchange",
"onemptied",
"onended",
"onerror",
"onfocus",
"oninput",
"oninvalid",
"onkeydown",
"onkeypress",
"onkeyup",
"onload",
"onloadeddata",
"onloadedmetadata",
"onloadstart",
"onmousedown",
"onmouseenter",
"onmouseleave",
"onmousemove",
"onmouseout",
"onmouseover",
"onmouseup",
"onmousewheel",
"onpause",
"onplay",
"onplaying",
"onprogress",
"onratechange",
"onreset",
"onresize",
"onscroll",
"onseeked",
"onseeking",
"onselect",
"onshow",
"onsort",
"onstalled",
"onsubmit",
"onsuspend",
"ontimeupdate",
"ontoggle",
"onvolumechange",
"onwaiting",
],
"supported": [
  "className",
  "dataset.*",
  "style.visibility"
]
},
"a": {
  "def": "href",
  "forbidden": [],
  "supported": [
    "href",
    "target"
  ]
}
}
```

```
    "innerHTML",
    "style.backgroundColor",
    "style.borderColor",
    "style.borderStyle",
    "style.borderWidth",
    "style.borderRadius",
    "style.color",
    "style.fontFamily",
    "style.fontSize",
    "style.fontStyle",
    "style.fontWeight",
    "style.textAlign",
    "style.textDecoration",
  ],
},
"button": {
  "def": "innerHTML",
  "forbidden": [],
  "supported": [
    "disabled",
    "innerHTML",
    "style.backgroundColor",
    "style.borderColor",
    "style.color",
    "style.fontFamily",
    "style.fontSize",
    "style.fontStyle",
    "style.fontWeight",
    "style.fontFamily",
    "style.textAlign",
  ],
},
"hr": {
  "def": "style.borderColor",
  "forbidden": [],
  "supported": [
    "style.backgroundColor"
    "style.borderColor"
    "style.borderWidth"
  ],
},
"img": {
  "def": "src",
  "forbidden": [],
  "supported": [
    "style.backgroundColor"
    "style.borderColor"
    "style.borderStyle"
    "style.borderWidth"
    "style.src"
  ],
},
"input": {
  "def": "value",
  "forbidden": [],
  "supported": [
    "disabled"
  ],
}
```

```
        "min",
        "max",
        "step",
        "style.backgroundColor",
        "style.borderColor",
        "style.borderStyle",
        "style.borderWidth",
        "style.color",
        "style.fontFamily",
        "style.fontSize",
        "style.fontStyle",
        "style.fontWeight",
        "style.textAlign",
        "value",
    ],
},
"label": {
    "def": "innerHTML",
    "forbidden": [],
    "supported": [
        "innerHTML",
        "style.backgroundColor",
        "style.borderColor",
        "style.borderStyle",
        "style.borderWidth",
        "style.color",
        "style.fontFamily",
        "style.fontSize",
        "style.fontStyle",
        "style.fontWeight",
        "style.textAlign",
    ]
},
"pre": {
    "def": "innerHTML",
    "forbidden": [],
    "supported": ["innerHTML"]
},
"range": {
    "def": "value",
    "forbidden": [],
    "supported": [
        "disabled",
        "min",
        "max",
        "step",
        "value",
    ]
},
"select": {
    "def": "value",
    "forbidden": [],
    "supported": [
        "disabled",
        "options",
        "selectedIndex",
        "style.backgroundColor",
    ]
},
```

```
    "style.borderColor",
    "style.borderWidth",
    "style.color",
    "style.fontFamily",
    "style.fontSize",
    "style.fontStyle",
    "style.textAlign",
    "value",
  ],
}
},
"writable_attributes": ["value"],
"xor_attributes": {
  "warning": [
    "selectedIndex", "value"
  ]
},
}
```

ABBILDUNG

| | |
|--|-----|
| Abhängigkeit der Tools | 143 |
| Absolute, konstante Ausrichtung | 164 |
| Andere Schulterkonfiguration möglich | 168 |
| Anfangs- und Endposition | 159 |
| Beziehung zwischen: frame / point / tool / trsf | 131 |
| Definition der Entferungen: ' leave ' / ' reach ' | 161 |
| Drehung des Koordinatensystems bezüglich folgender Achse: X | 136 |
| Drehung des Koordinatensystems bezüglich folgender Achse: Y | 137 |
| Drehung des Koordinatensystems bezüglich folgender Achse: Z | 137 |
| Geradlinige Bewegung | 159 |
| Konfiguration: enegative | 153 |
| Konfiguration: epositive | 153 |
| Konfiguration: lefty | 152 |
| Konfiguration: lefty | 155 |
| Konfiguration: righty | 152 |
| Konfiguration: righty | 155 |
| Konfiguration: wnegative | 154 |
| Konfiguration: wpositive | 154 |
| Konstante Orientierung bezüglich der Bahn | 164 |
| Kreisförmige Bewegung | 160 |
| Mehrdeutigkeit der mittleren Orientierung | 165 |
| Orientierung | 136 |
| Positive / negative Konfiguration des Ellenbogens | 166 |
| Positive / negative Konfiguration des Handgelenks | 167 |
| Punktdefinition | 146 |
| Sequentielles Ordnen | 90 |
| Vollständiger Kreis | 165 |
| Wechsel: righty / lefty | 166 |
| Wechseln der Ellenbogenkonfiguration nicht möglich | 167 |
| Zusammenhang zwischen den Referenz-Koordinatensystemen | 141 |
| Zwei verschiedene Konfigurationen, um denselben Punkt zu erreichen: P | 150 |
| Zyklus mit Blending | 162 |
| Zyklus mit Blending | 169 |
| Zyklus ohne Blending an einem Punkt | 162 |
| Zyklus: U | 160 |

INDEX

A

abs (Funktion) 47, 132
accel 172
acos (Funktion) 46
aio 28, 63
aioGet (Funktion) 63
aioLink (Funktion) 63
aioSet (Funktion) 64
align (Funktion) 140
alter (Funktion) 198
alterBegin (Funktion) 197
alterEnd (Funktion) 198
alterMovec (Funktion) 196
alterMovej (Funktion) 195
alterMovel (Funktion) 196
alterStopTime (Funktion) 199
append (Funktion) 35
appro (Funktion) 148
asc (Funktion) 57
asin (Funktion) 46
atan (Funktion) 47
autoConnectMove 163
autoConnectMove (Funktion) 179

B

bAnd (Funktion) 50
blend 161, 172
bNot (Funktion) 50
bool 28
bOr (Funktion) 51
bXor (Funktion) 51

C

call (Funktion) 22
chr (Funktion) 56
clearBuffer (Funktion) 69
clock (Funktion) 103
close 90
close (Funktion) 145
codeAscii 56
compose (Funktion) 147
config 28, 131, 150
config (Funktion) 155
cos (Funktion) 46

D

decel 172
delay 90
delay (Funktion) 102
delete (Funktion) 33, 58
dio 28, 59
dioGet (Funktion) 60
dioLink (Funktion) 60
dioSet (Funktion) 61
disablePower 90
disablePower (Funktion) 121

distance (Funktion) 138, 147
do ... until (Funktion) 24

E

elbow 150
enablePower (Funktion) 121
enegative 153
epositive 153
esStatus (Funktion) 122
exp (Funktion) 47

F

find (Funktion) 58
first (Funktion) 36
for (Funktion) 25
frame 28, 131
fromBinary (Funktion) 52

G

getData (Funktion) 34
getDate 85
getDefaultValue (Funktion) 202
getDH (Funktion) 202
getEvents (Funktion) 189
getIds (Funktion) 189
getJntRef (Funktion) 124
getJointForce (Funktion) 179
getLanguage (Funktion) 84
getLatch (Funktion) 134
getLicence (Funktion) 200
getMonitorSpeed (Funktion) 123
getMoveld (Funktion) 180
getPayload (Funktion) 186
getPositionErr (Funktion) 179
getProfile (Funktion) 83
getSpeed (Funktion) 179
getVersion (Funktion) 124
globale 30

H

hasCpuExtPowerSupply (Funktion) 126, 127
help (Funktion) 98
here (Funktion) 148
herej (Funktion) 133
hibernateRobot (Funktion) 125

I

if (Funktion) 23
insert (Funktion) 32, 58
interpolateC (Funktion) 140
interpolateL (Funktion) 139
ioStatus (Funktion) 61, 62, 64, 65, 69
isCalibrated (Funktion) 121
isDefined (Funktion) 31
isEmpty (Funktion) 178
isInRange (Funktion) 133

isPowered (Funktion) 121
isSettled (Funktion) 178

J

joint 28
jointToPoint (Funktion) 148

L

last (Funktion) 36
leave 161, 172
left (Funktion) 57
lefty 152, 155
len (Funktion) 58
libDelete (Funktion) 111
libExist (Funktion) 112
libList (Funktion) 112
libLoad 110
libLoad (Funktion) 111
libPath (Funktion) 112
libSave (Funktion) 111
limit (Funktion) 49
link (Funktion) 142, 145, 149
In (Funktion) 48
locale 30
log (Funktion) 48
logMsg (Funktion) 82

M

mainPowerOkFailure (Funktion) 126
max (Funktion) 49
mdesc 28, 159, 172
mid (Funktion) 57
min (Funktion) 49
movec (Funktion) 175
movej 159
movej (Funktion) 173
movel 159
movel (Funktion) 174

N

next (Funktion) 36
num 28

O

open 90
open (Funktion) 144

P

point 28
pointToJoint (Funktion) 149
popUpMsg (Funktion) 82
position (Funktion) 142, 145, 149
power (Funktion) 47
prev (Funktion) 36

R

reach 161, 172
replace (Funktion) 58
resetMotion 163, 176
resetMotion (Funktion) 176
resetTurn (Funktion) 203
resize (Funktion) 35
restartMove 176
restartMove (Funktion) 177
return (Funktion) 22
right (Funktion) 57
righty 152, 155
round (Funktion) 49
roundDown (Funktion) 48
roundUp (Funktion) 48
RUNNING 102
rvel 172

S

sel (Funktion) 49
setDH (Funktion) 202
setFrame (Funktion) 142
setLanguage (Funktion) 84
setLatch (Funktion) 134
setMonitorSpeed (Funktion) 123
setMoveld (Funktion) 180
setMutex 90
setMutex (Funktion) 98
setPayload (Funktion) 185
setProfile (Funktion) 83
shoulder 150
sin (Funktion) 46
sio 28, 66
sioCtrl (Funktion) 71
sioGet 90
sioGet (Funktion) 70
sioLink (Funktion) 69
sioSet 90
size (Funktion) 31, 35
sqrt (Funktion) 47
stopMove (Funktion) 176
STOPPED 97
string 28
switch (Funktion) 26

T

tan (Funktion) 46
taskCreate (Funktion) 100
taskCreateSync (Funktion) 101
taskKill 90
taskKill (Funktion) 98
taskResume 89, 90
taskResume (Funktion) 97
taskStatus 89
taskStatus (Funktion) 99
taskSuspend (Funktion) 97
toBinary (Funktion) 52

toNum (Funktion) 55

tool 28, 131

toString (Funktion) 54

trsF 28, 131

tvel 172

U

UserPage (Funktion) 75

userPageAlert (Funktion) 81

userPageBind (Funktion) 76

userPageBindXEVENTX (Funktion) 78

userPageConfirm (Funktion) 81

userPagePrompt (Funktion) 81, 82

userPageUnbind (Funktion) 77

userPageUnbindXEVENTX (Funktion) 79

V

vel 172

W

wait 90

wait (Funktion) 102

waitEndMove 90, 162

waitEndMove (Funktion) 177

wakeUpRobot (Funktion) 125

watch 90

watch (Funktion) 103

wdgRefresh (Funktion) 96

while (Funktion) 24

wnegative 154

workingMode (Funktion) 122

wpositive 154

wrist 150

