

www.g2robotics.de

g2Framework
KUKA System Software

Jul 11, 2025

Inhaltsverzeichnis

1 Einführung.....	1
1.1 Keine Garantie.....	1
1.2 Fehler melden.....	1
2 Lizenz.....	2
3 Voraussetzungen.....	3
4 Installation.....	4
4.1 KRL-Bibliotheken.....	4
4.2 g2Framework.kop.....	4
4.2.1 Installation auf der Robotersteuerung.....	4
4.2.2 Installation mit WorkVisual (empfohlen).....	5
5 Systemvariablen.....	6
5.1 Cycflags.....	6
5.2 Interrupts - Submitinterpreter.....	6
5.3 Interrupts - Roboterinterpreter.....	6
6 Bibliotheken.....	7
6.1 MathLib: Mathematische Funktionen.....	7
6.1.1 Prüfung auf gerade und ungerade Zahlen.....	7
6.1.1.1 mal_isEven(...).....	7
6.1.1.2 mal_isOdd(...).....	8
6.1.2 Maximum und Minimum Funktionen.....	8
6.1.2.1 mal_maxInt(...).....	8
6.1.2.2 mal_minInt(...).....	9
6.1.2.3 mal_maxReal(...).....	9
6.1.2.4 mal_minReal(...).....	10
6.1.3 Testen auf Gleichheit und Intervall.....	11
6.1.3.1 mal_realComp(...).....	11
6.1.3.2 mal_validationRange (...).....	11
6.1.3.3 mal_validationRelat(...).....	12
6.1.4 Rundungsfunktionen.....	13
6.1.4.1 mal_floor(...).....	13
6.1.4.2 mal_ceil(...).....	14
6.1.4.3 mal_trunc(...).....	15
6.1.5 Elementare Funktionen.....	15
6.1.5.1 mal_modulo (...).....	15
6.1.5.2 mal_2pow (...).....	16
6.1.5.3 mal_powInt (...).....	17
6.1.5.4 mal_powUInt (...).....	17
6.1.5.5 mal_hypot (...).....	18
6.1.5.6 mal_hypot3(...).....	18
6.1.6 Bit-Operationen.....	19
6.1.6.1 mal_bitState(...).....	19
6.1.6.2 mal_shiftBitsRight(...).....	20
6.1.7 Sortieralgorithmen.....	20
6.1.7.1 mal_quickSort(...).....	20
6.1.8 Vektoren.....	21
6.1.8.1 mal_vectorLength(...).....	21

6.1.8.2 mal_vectorProduct(...)	22
6.1.8.3 mal_tripleProduct(...)	22
6.1.8.4 mal_vectorAngle(...)	23
6.1.8.5 mal_vectorMinMax(...)	24
6.1.8.6 mal_vectorAverage(...)	24
6.1.8.7 mal_vectorAvgMinMax (...)	25
6.1.8.8 mal_vectorNormalization (...)	26
6.1.9 Matrizen.....	26
6.1.9.1 mal_rotMatToRPY.....	27
6.1.10 Geometrie.....	27
6.1.10.1 mal_circumference(...)	28
6.1.10.2 mal_circularArc(...)	28
6.1.11 Zufallszahlen.....	29
6.1.11.1 mal_randLcg(...)	29
6.1.11.2 mal_randLcgRange(...)	29
6.2 PhysLib: Physikalische Funktionen.....	30
6.2.1 Temperatur.....	30
6.2.1.1 phl_linThermalExpansion(...)	30
6.2.1.2 phl_CelsiusToKelvin(...)	31
6.2.1.3 phl_KelvinToCelsius(...)	31
6.2.1.4 phl_CelsiusToFahrenheit(...)	32
6.2.1.5 phl_FahrenheitToCelsius(...)	32
6.2.1.6 phl_KelvinToFahrenheit(...)	33
6.2.1.7 phl_FahrenheitToKelvin(...)	33
6.2.2 Zeit.....	34
6.2.2.1 phl_isLeapYear(...)	34
6.2.2.2 phl_daysInMonth(...)	34
6.2.2.3 phl_daysInYear(...)	35
6.2.2.4 phl_daysElapsedOfYear(...)	35
6.2.2.5 phl_secInYear(...)	36
6.2.2.6 phl_secElapsedOfYear(...)	36
6.2.2.7 phl_secSince2000(...)	37
6.2.2.8 phl_timeSpan(...)	37
6.2.2.9 phl_timeSpanDays(...)	38
6.2.2.10 phl_dateCompare(...)	39
6.2.2.11 phl_dateStamp(...)	39
6.2.2.12 phl_timeStamp(...)	40
6.3 RobLib: Roboterspezifische Funktionen.....	40
6.3.1 Transformation.....	40
6.3.1.1 rol_ForwardTransform(...)	41
6.3.1.2 rol_BackwardTransform(...)	41
6.3.1.3 rol_BaseFromPositions(...)	42
6.3.2 Testen auf Gleichheit.....	43
6.3.2.1 rol_FrameCompare(...)	43
6.3.3 Distanzen.....	44
6.3.3.1 rol_FrameDistance(...)	44
6.3.3.2 rol_cartPointDistance(...)	44
6.3.4 Orientierungen.....	45
6.3.4.1 rol_toolOriToGravity(...)	45
6.3.4.2 rol_posDirToGravity(...)	45

6.3.5 Timerfunktion.....	46
6.3.5.1 rol_elapsedMsecRobTimer(...)	46
6.3.6 Roboter-Halt.....	47
6.3.6.1 rol_robStopAndRelease(...)	47
6.3.6.2 rol_robStop(...)	47
6.3.6.3 rol_robStopRelease(...)	48
6.3.7 Trace.....	48
6.3.7.1 rol_TraceStart(...)	48
6.3.7.2 rol_TraceEnd()	49
6.4 AxisLib: Achsspezifische Funktionen.....	49
6.4.1 Positionen.....	49
6.4.1.1 axl_axisCmdPos(...)	49
6.4.1.2 axl_axisMeasPos(...)	50
6.4.1.3 axl_axisMot(...)	50
6.4.1.4 axl_axisPosInfo(...)	51
6.4.1.5 axl_exaxPosE6Pos(...)	52
6.4.1.6 axl_setExAxValueE6pos(...)	52
6.4.1.7 axl_setAxisValue(...)	53
6.4.1.8 axl_getAxisValue(...)	54
6.4.1.9 axl_setE6axisElement(...)	54
6.4.2 Distanzen.....	55
6.4.2.1 axl_axisDistPosToPos(...)	55
6.4.2.2 axl_e6AxisDistToDestPos(...)	56
6.4.2.3 axl_exAxisDistToDestPos(...)	56
6.4.2.4 axl_e6AxisDist(...)	57
6.4.2.5 axl_axisDist(...)	58
6.4.3 Geschwindigkeiten.....	58
6.4.3.1 axl_maxVelOutput(...)	58
6.4.3.2 axl_absVelToRelVel(...)	59
6.4.3.3 axl_relVelToAbsVel(...)	59
6.4.3.4 axl_axisVelInfo(...)	60
6.4.4 Momente.....	61
6.4.4.1 axl_axisTorqueInfo(...)	61
6.4.5 Übersetzung.....	61
6.4.5.1 axl_linRatioSpindleDrive(...)	61
6.4.6 Achsenzustand.....	62
6.4.6.1 axl_axisMastered(...)	62
6.4.6.2 axl_brakeOpen(...)	63
6.4.6.3 axl_exAxIsAsynchron(...)	63
6.4.6.4 axl_exAxSynchron(...)	64
6.4.6.5 axl_exAxAsynchron(...)	64
6.4.6.6 axl_exAxIsCoupled(...)	65
6.4.6.7 axl_exAxCouple(...)	65
6.4.6.8 axl_exAxDecouple(...)	66
6.4.7 Bewegungen.....	66
6.4.7.1 axl_exAxAsyncMovement(...)	66
6.5 MsgLib: Anwendermeldungen.....	67
6.5.1 Meldungen abfeuern.....	67
6.5.1.1 msl_FireMsg()	67
6.5.1.2 msl_FireDialog()	69

6.5.1.3 msl_FireDialogQuitExt().....	70
6.5.1.4 msl_FireStringParams().....	72
6.5.1.5 msl_FireFrameData().....	73
6.5.2 Meldungszustand.....	74
6.5.2.1 msl_ExistsMsg(...).....	74
6.5.2.2 msl_ExistsDialog(..).....	75
6.5.2.3 msl_Buffer(..).....	75
6.5.3 Meldungen löschen.....	76
6.5.3.1 msl_Clear(..).....	76
6.5.4 Parameterzuweisung.....	77
6.5.4.1 msl_paramInt(..).....	77
6.5.4.2 msl_paramReal(..).....	77
6.5.4.3 msl_paramBool(..).....	78
6.5.4.4 msl_paramString(..).....	78
6.6 FileLib: Dateien.....	79
6.6.1 Daten auf Festplatte schreiben.....	79
6.6.1.1 fil_LogValuesToCsv(...).....	79
6.6.1.2 fil_LogLineToCsv(...).....	80
6.6.1.3 fil_LogLineToTxt(...).....	81
6.6.1.4 fil_LogMessage(...).....	81
6.6.2 Dateibearbeitung.....	82
6.6.2.1 fil_FileExists(...).....	82
6.6.2.2 fil_FileRemove(...).....	83
6.7 StringLib: Zeichenketten.....	83
6.7.1 Variablenwerte konvertieren.....	83
6.7.1.1 stl_boolToString(..).....	83
6.7.1.2 stl_intToString(..).....	84
6.7.1.3 stl_realToString(..).....	84
6.7.1.4 stl_frameToString(..).....	85
6.7.1.5 stl_axisToString(..).....	86
6.7.1.6 stl_dateToString(..).....	87
6.7.1.7 stl_timeToString(..).....	87
6.7.1.8 stl_dateTimeToString(..).....	88
6.7.1.9 stl_stringToInt(..).....	89
6.7.1.10 stl_padLeadingZeros(...).....	89
6.7.2 Zeichenketten bearbeiten.....	90
6.7.2.1 stl_toUpper(..).....	90
6.7.2.2 stl_toLower(..).....	91
6.7.2.3 stl_replaceString (..).....	91
6.7.2.4 stl_replaceParams(..).....	92
6.7.2.5 stl_Trim(..).....	93
6.7.2.6 stl_TrimLeft(..).....	93
6.7.2.7 stl_TrimRight(..).....	94
6.7.2.8 stl_subString(..).....	95
6.7.2.9 stl_splitString(..).....	95
6.7.2.10 stl_findString(..).....	96
6.7.3 Zeichenketten prüfen.....	98
6.7.3.1 stl_charIsNumeric (..).....	98
6.7.3.2 stl_charIsAlphaNumeric (..).....	98
6.7.3.3 stl_charIsLowerCase (..).....	99

6.7.3.4	stl_charIsUpperCase (..)	99
6.7.3.5	stl_stringIsNullOrSpace (...)	100
6.7.3.6	stl_ipAdressValid(...)	101
6.8	Varlib: Variablen und Typen	101
6.8.1	Typenkonvertierung	102
6.8.1.1	val_UintToInt(...)	102
6.8.1.2	val_IntToUint(...)	102
6.8.1.3	val_bytesToInt(...)	103
6.8.1.4	val_shiftBitsRight(...)	103
6.8.2	Bit-Konvertierung	104
6.8.2.1	val_bitConvIntToBuffer(...)	104
6.8.2.2	val_bitConvRealToBuffer(...)	104
6.8.2.3	val_bitConvBoolToBuffer(...)	105
6.8.2.4	val_bitConvBufferToInt(...)	105
6.8.2.5	val_bitConvBufferToReal(...)	105
6.8.2.6	val_bitConvBufferToBool(...)	106
6.8.2.7	val_bitConvRealToInt(...)	106
6.8.2.8	val_bitConvIntToReal(...)	106
6.8.2.9	val_bitConvBufferCopy(...)	107
6.8.3	Byte-swapping	107
6.8.3.1	val_byteSwap(...)	107
6.8.3.2	val_byteSwapInteger(...)	108
6.8.4	Daten Validierung	109
6.8.4.1	val_signalOverflow(...)	109
6.8.4.2	val_validationParamRange(...)	109
6.8.4.3	val_validationParamRelat(...)	110
7	Interpreter-Ereignisse	112
7.1	KRL-Module	112
7.2	Ereignisübersicht	112
7.3	Submit-Diagnose	113
8	Visualisierung	114
8.1	ApplicationView	114
8.1.1	Funktionalität	114
8.1.2	Konfiguration	115
8.1.2.1	Applikationsdefinition	115
8.1.2.2	Beschreibungsdatei	117
8.1.2.3	Variablendeklaration in KRL	118
8.1.3	Bedienung	118
8.1.3.1	Aufruf	118
8.1.3.2	Variablenwert ändern	118
8.1.3.3	Variablenwert zyklisch aktualisieren	119
8.1.3.4	Variablenbeschreibung	119

1 Einführung

Das g2Framework ist eine Sammlung von KRL-Routinen sowie einem Plugin zur Konfiguration von Applikationen.

Die Routinen decken einen grossen Bereich von Funktionen der Robotersteuerung ab.

Die Verwendung der Routinen und Plugins werden in dieser Dokumentation beschrieben.

1.1 Keine Garantie

Für die in diesem Dokument beschriebene Software besteht keine Garantie. Es liegt in der Verantwortung des Benutzers, die Routinen auf ihre Funktionalität und Sicherheit zu prüfen. Weitere Informationen sind der BSD-Lizenz zu entnehmen.

1.2 Fehler melden

Sollten bei der Benutzung der Routinen Fehler auftreten, melden Sie diese bitte an **info@g2robotics.de**.

Der Fehlerreport sollte folgende Informationen beinhalten:

- Version des KRL-Moduls
(Hauptmenü- → Anzeige → g2Framework → ApplicationView:
Applikation: g2Framework, Komponente: g2GlobalLibs)
- Version der KUKA-Basissoftware
- Eine Beschreibung des Fehlers
- Ein Programmbeispiel zur Fehlernachstellung

2 Lizenz

By downloading, copying, installing or using the software you agree to this license. If you do not agree to this license, do not download, install, copy or use the software.

**License Agreement
g2Framework
(3-clause BSD License)**

*Copyright (C) 2019-2020, www.g2robotics.de, all rights reserved.
Third party copyrights are property of their respective owners.*

*The Regents of the University of California. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:*

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

3 Voraussetzungen

Die Routinen und Funktionalitäten werden unter KSS8.6 erstellt und getestet.

Folgende Plattformen werden unterstützt:

- KSS8.6 (KUKA System Software)
- KSS8.7 (KUKA System Software)
- VSS8.6 (Volkswagen System Software)
- VSS8.7 (Volkswagen System Software)

4 Installation

4.1 KRL-Bibliotheken

Die KRL-Bibliotheken „g2GlobalLibs“ können auf der Robotersteuerung in das Verzeichnis \R1 kopiert werden. Abhängig von der Plattform erfordert der Kopiervorgang die Benutzergruppe „Experte“.

Alternativ können die Bibliotheken über „WorkVisual“ auf die Steuerung kopiert werden:

Vorgehensweise:

- Netzwerkverbindung zwischen WorkVisual und der Robotersteuerung.
- WorkVisual → Arbeitsbereich: Programmierung und Diagnose.
- Zellenansicht: Steuerung selektieren.
- KRC Explorer: Mausklick rechts auf die Steuerung.
- Kontextmenü: „Steuerungsstand herstellen“.
- Module „g2GlobalLibs“ per drag & drop in das \R1 – Verzeichnis ziehen.
- KRC Explorer: Mausklick rechts.
- Kontextmenü: „Änderungen übertragen“.

4.2 g2Framework.kop

4.2.1 Installation auf der Robotersteuerung

Voraussetzung:

- g2Framework.kop auf einem USB-Stick an der Robotersteuerung.
- Benutzergruppe „Experte“

Vorgehensweise:

- Hauptmenü: Inbetriebnahme → Zusatzsoftware.
- Button: „Neue Software“.
- Button: „Konfigurieren“.
- Button „Pfadauswahl“.
- USB-Stick selektieren und Button „Speichern“.
- Button „Speichern“.
- „InstallTech-Auswahl“ g2Framework selektieren.
- Button „Installieren“.

4.2.2 Installation mit WorkVisual (empfohlen)

Voraussetzung:

- Netzwerkverbindung zwischen WorkVisual und der Robotersteuerung.
- Benutzergruppe „Experte“.

Vorgehensweise:

- Installation g2Framework.kop in WorkVisual (Dokumentation KUKA).
- Aktuelles Projekt von der Robotersteuerung laden.
- g2Framework in das Projekt unter „Optionen“ einfügen.
- Projekt auf die Robotersteuerung übertragen und aktivieren.

5 Systemvariablen

Übersicht der im g2Framework verwendeten Systemvariablen.

5.1 Cycflags

Nummer	Name	Deklaration	Beschreibung
160	val_cy_msgLibWait	g2_varLib.dat	Bedingung für Wartemeldung.
161	val_cy_edgeMonitor	g2_varLib.dat	Überwachung im Roboterinterpreter

5.2 Interrupts - Submitinterpreter

Prio	Name	Deklaration	Beschreibung
101	val_int_subLineSel	g2_varLib.dat	Interrupt bei Satzanwahl
102	val_int_subInitHot	g2_varLib.dat	Interrupt nach Warmstart des Submitinterpreter

5.3 Interrupts - Roboterinterpreter

Prio	Name	Deklaration	Beschreibung
100	val_int_robProgStart	g2_varLib.dat	Interrupt bei Programmstart (\$PRO_STATE1==#P_ACTIVE)
101	val_int_robCycRising	g2_varLib.dat	Interrupt bei steigender Flanke \$CYCFLAG[val_cy_edgeMonitor]
102	val_int_robCycFalling	g2_varLib.dat	Interrupt bei fallender Flanke \$CYCFLAG[val_cy_edgeMonitor]

6 Bibliotheken

Übersicht:

Bibliothek	Beschreibung
g2_mathLib	Mathematische Funktionen
g2_physLib	Physikalische Funktionen
g2_robLib	Roboterspezifische Funktionen
g2_axisLib	Achsspezifische Funktionen
g2_msgLib	Meldungsausgaben
g2_fileLib	Schreiben von Daten oder Text in Dateien
g2_stringLib	Bearbeitung von Zeichenketten
g2_varLib	Behandlung von Variablen und Typen

6.1 MathLib: Mathematische Funktionen

6.1.1 Prüfung auf gerade und ungerade Zahlen

6.1.1.1 *mal_isEven(...)*

Beschreibung:

Prüfung ob eine Ganzzahl gerade ist.

Funktionskopf:

GLOBAL DEFFCT BOOL mal_isEven(n:IN)

Rückgabe:

- True: Wert ist gerade.
- False: Wert ist ungerade.

Parameter:

Parameter	Datentyp	Beschreibung
n	INT	Wert der geprüft wird, ob er gerade ist.

Beispiel:

```
INI intValue
BOOL bReturn

intValue=8
bReturn = mal_isEven(intValue)
```

```
//bReturn==TRUE

intValue=45367
bReturn = mal_isEven(intValue)

//bReturn==FALSE
```

6.1.1.2 **mal_isOdd(...)**

Beschreibung:

Prüfung ob eine Ganzzahl ungerade ist.

Funktionskopf:

GLOBAL DEFFCT BOOL mal_isOdd(n:IN)

Rückgabe:

- True: Wert ist ungerade.
- False: Wert ist gerade.

Parameter:

Parameter	Datentyp	Beschreibung
n	INT	Wert der geprüft wird, ob er ungerade ist.

Beispiel:

```
INT intValue
BOOL bReturn

intValue=8
bReturn = mal_isOdd(intValue)

//bReturn==FALSE

intValue=45367
bReturn = mal_isOdd(intValue)

//bReturn==TRUE
```

6.1.2 Maximum und Minimum Funktionen

6.1.2.1 **mal_maxInt(...)**

Beschreibung:

Bestimmung des Maximums zweier ganzer Zahlen.

Funktionskopf:

GLOBAL DEFFCT INT mal_maxInt(n1:IN, n2:IN)

Rückgabe:

Maximum zweier integer-Zahlen.

Parameter:

Parameter	Datentyp	Beschreibung
n1	INT	Vergleichswert 1
n2	INT	Vergleichswert 2

Beispiel:

```
INT iValue1, iValue2, iMax
```

```
iValue1=-541  
iValue2=672
```

```
iMax=mal_maxInt(iValue1, iValue2)
```

```
//iMax==672
```

6.1.2.2 *mal_minInt(...)*

Beschreibung:

Bestimmung des Minimums zweier ganzer Zahlen.

Funktionskopf:

GLOBAL DEFFCT INT mal_minInt(n1:IN, n2:IN)

Rückgabe:

Minimum zweier integer-Zahlen.

Parameter:

Parameter	Datentyp	Beschreibung
n1	INT	Vergleichswert 1
n2	INT	Vergleichswert 2

Beispiel:

```
INT iValue1, iValue2, iMin
```

```
iValue1=-541  
iValue2=672
```

```
iMin=mal_minInt(iValue1, iValue2)
```

```
//iMin==-541
```

6.1.2.3 *mal_maxReal(...)*

Beschreibung:

Bestimmung des Maximums zweier reeller Zahlen.

Funktionskopf:

GLOBAL DEFFCT REAL mal_maxReal(x1:IN, x2:IN)

Rückgabe:

Maximum zweier Real-Zahlen.

Parameter:

Parameter	Datentyp	Beschreibung
x1	REAL	Vergleichswert 1
x2	REAL	Vergleichswert 2

Beispiel:

```
REAL rValue1, rValue2, rMax

rValue1=41.654
rValue2=67.256

rMax=mal_maxReal(rValue1, rValue2)

//rMax==67.256
```

6.1.2.4 mal_minReal(...)

Beschreibung:

Bestimmung des Minimums zweier reeller Zahlen.

Funktionskopf:

GLOBAL DEFFCT REAL mal_minReal(x1:IN, x2:IN)

Rückgabe:

Minimum zweier Real-Zahlen.

Parameter:

Parameter	Datentyp	Beschreibung
x1	REAL	Vergleichswert 1
x2	REAL	Vergleichswert 2

Beispiel:

```
REAL rValue1, rValue2, rMin

rValue1=41.654
rValue2=-67.256

rMin=mal_minReal(rValue1, rValue2)

//rMin==-67.256
```

6.1.3 Testen auf Gleichheit und Intervall

6.1.3.1 *mal_realComp(...)*

Beschreibung:

Diese Funktion bestimmt, ob x und y ungefähr gleich einer relativen Genauigkeit sind. Wenn x und y innerhalb dieses Intervalls liegen, werden sie als ungefähr gleich angesehen und die Funktion gibt TRUE zurück. Andernfalls gibt die Funktion FALSE zurück. Wird kein Wert für die Intervallgrenze übergeben, rechnet die Funktion mit einer Grenze von 6E-8.

Funktionskopf:

GLOBAL DEFFCT BOOL mal_realComp(x1:IN, x2:IN, comp:IN)

Rückgabe:

- True: Die Differenz der reellen Zahlen x1, x2 liegt innerhalb des Intervalls (0...comp).
- False: Die Differenz der reellen Zahlen x1, x2 liegt ausserhalb des Intervalls (0...comp).

Parameter:

Parameter	Datentyp	Beschreibung
x1	REAL	Vergleichswert 1
x2	REAL	Vergleichswert 2
comp	REAL	Intervallgrenze

Beispiel:

```
REAL x1, x2, comp
BOOL isEqual

x1=41.654
x2=41.526
comp=0.2

isEqual=mal_realComp(x1, x2, comp)
//isEqual==TRUE

;compare difference of x1, x2 with boundary of 6E8
isEqual=mal_realComp(x1, x2, )
//isEqual==FALSE
```

6.1.3.2 *mal_validationRange (...)*

Beschreibung:

Validierung einer Zahl auf Intervallgrenzen.

Bei einer Prüfung innerhalb von Grenzen gilt das abgeschlossene Intervall:

→ (lowerBoundary <= value <=upperBoundary).

Bei einer Prüfung außerhalb von Grenzen gilt das offene Intervall:

→ (value < lowerBoundary) ODER (value > upperBoundary).

Funktionskopf:

GLOBAL DEFFCT BOOL mal_validationRange(value:IN, lowerBoundary:IN, upperBoundary:IN, location:IN)

Rückgabe:

- True: Es liegt keine Bereichsverletzung vor.
- False: Es liegt eine Bereichsverletzung vor.

Parameter:

Parameter	Datentyp	Beschreibung
value	Real	Zahl zur Prüfung
lowerBoundary	Real	Untere Grenze des Intervalls. (lowerBoundary<=upperBoundary)
upperBoundary	Real	Obere Grenze des Intervalls (upperBoundary>=lowerBoundary)
location	Enum	Definition, ob die Zahl innerhalb oder außerhalb des Intervalls geprüft werden soll. • #inside: Prüfung, ob sich die Zahl innerhalb des Intervalls befindet. • #outside: Prüfung, ob sich die Zahl außerhalb des Intervalls befindet.

Beispiel:

```
INT value, lowerBoundary, upperBoundary
DECL val_Location_T location
BOOL inRange

value=123
LowerBoundary=45
UpperBoundary=614
location=#inside

inRange = mal_validationRange(value, lowerBoundary, upperBoundary, location)

//inRange==TRUE; lowerBoundary <= value <=upperBoundary

location=#outside
inRange = mal_validationRange(value, lowerBoundary, upperBoundary, location)

//inRange==FALSE; lowerBoundary <= value <=upperBoundary
```

6.1.3.3 mal_validationRelat(...)

Beschreibung:

Validierung einer Zahl gegen einen Vergleichswert.

Funktionskopf:

GLOBAL DEFFCT BOOL mal_validationRelat(value:IN, relOperator:IN, compValue:IN)

Rückgabe:

- True: Bedingung ist erfüllt.
- False: Bedingung ist nicht erfüllt.

Parameter:

Parameter	Datentyp	Beschreibung
value	Real	Zahl zur Prüfung
relOperator	Enum	Vergleichsoperator • #EQUAL, Prüfung auf Gleichheit • #UNEQUAL, Prüfung auf Ungleichheit • #GREATER, Prüfung auf größer • #LESS, Prüfung auf kleiner • #GREATEREQUAL, Prüfung auf größer gleich • #LESSEQUAL, Prüfung auf kleiner gleich
compValue	Real	Vergleichswert

Beispiel:

```

REAL rValue, compValue
DECL val_RelationalOperator_T relOperator
BOOL isValid

rValue=123.654
compValue=654.321
relOperator=#LESS

isValid = mal_validationRelat(rValue, relOperator, compValue)

//isValid==TRUE;  rValue <  compValue

```

6.1.4 Rundungsfunktionen

6.1.4.1 *mal_floor(...)*

Beschreibung:

Die Funktion ermittelt die größte Ganzzahl, die kleiner oder gleich x ist.

Funktionskopf:

GLOBAL DEFFCT INT mal_floor(x :IN)

Rückgabe:

X auf die nächst kleinere Ganzzahl abgerundet.

Parameter:

Parameter	Datentyp	Beschreibung
x	REAL	Wert der auf Ganzzahl abgerundet wird. (-2147483648.0...2147483647.0)

Beispiel:

```

INT idx, iResult
INT value[6]
value[1]=7.03
value[2]=7.64
value[3]=0.12
value[4]=-0.12
value[5]=-7.1
value[6]=-7.6

FOR i= 1 TO 6
iResult=mal_floor(value[idx])
  msl_FireMsg(#NOTIFY, "g2_mathlib", 10060, "Example mal_floor, return:%1, value:%2",
    val_st_notMsgOpt, val_i_msgHandle,iResult,,value[idx],)
ENDFOR

//output
//Example mal_floor, return:7, value:7.03"
//Example mal_floor, return:7, value:7.64"
//Example mal_floor, return:0, value:0.12"
//Example mal_floor, return:-1, value:-0.12"
//Example mal_floor, return:-8, value:-7.1"
//Example mal_floor, return:-8, value:-7.6"

```

6.1.4.2 mal_ceil(...)

Beschreibung:

Die Funktion ermittelt die kleinste Ganzzahl, der groesser oder gleich x ist.

Funktionskopf:

GLOBAL DEFFCT INT mal_ceil(x :IN)

Rückgabe:

X auf die nächst größere Ganzzahl aufgerundet.

Parameter:

Parameter	Datentyp	Beschreibung
x	REAL	Wert der auf Ganzzahl aufgerundet wird. (-2147483648.0...2147483647.0)

Beispiel:

```

INT idx, iResult
INT value[6]
value[1]=7.03
value[2]=7.64
value[3]=0.12
value[4]=-0.12
value[5]=-7.1
value[6]=-7.6

FOR i= 1 TO 6
iResult=mal_ceil(value[idx])
  msl_FireMsg(#NOTIFY, "g2_mathlib", 10060, "Example mal_ceil, return:%1, value:%2",
    val_st_notMsgOpt, val_i_msgHandle,iResult,,value[idx],)
ENDFOR

//output
//Example mal_ceil, return:8, value:7.03"
//Example mal_ceil, return:8, value:7.64"
//Example mal_ceil, return:1, value:0.12"

```

```
//Example mal_ceil, return:0, value:-0.12"
//Example mal_ceil, return:-7, value:-7.1"
//Example mal_ceil, return:-7, value:-7.6"
```

6.1.4.3 *mal_trunc(...)*

Beschreibung:

Rundet auf die nächste Ganzzahl in Richtung 0

Funktionskopf:

GLOBAL DEFFCT INT mal_trunc(x :IN)

Rückgabe:

Ganzzahl ohne Dezimalstelle von x

Parameter:

Parameter	Datentyp	Beschreibung
x	REAL	Wert der auf die nächste Ganzzahl Richtung 0 gerundet wird. (-2147483648.0...2147483647.0)

Beispiel:

```
INT idx, iResult
INT value[6]
value[1]=7.03
value[2]=7.64
value[3]=0.12
value[4]=-0.12
value[5]=-7.1
value[6]=-7.6

FOR i= 1 TO 6
iResult=mal_trunc(value[idx])
  msl_FireMsg(#NOTIFY, "g2_mathlib", 10060, "Example mal_trunc, return:%1, value:%2",
    val_st_notMsgOpt, val_i_msgHandle,iResult,,value[idx],,)
ENDFOR

//output
//Example mal_trunc, return:7, value:7.03"
//Example mal_trunc, return:7, value:7.64"
//Example mal_trunc, return:0, value:0.12"
//Example mal_trunc, return:0, value:-0.12"
//Example mal_trunc, return:-7, value:-7.1"
//Example mal_trunc, return:-7, value:-7.6"
```

6.1.5 Elementare Funktionen

6.1.5.1 *mal_modulo (...)*

Beschreibung:

Berechnet den Rest aus einer Division zweier ganzer Zahlen.

Funktionskopf:

GLOBAL DEFFCT INT mal_modulo(numerator:IN, denominator:IN)

Rückgabe:

Rest einer Division zweier ganzer Zahlen.

Parameter:

Parameter	Datentyp	Beschreibung
numerator	INT	Dividend
denominator	INT	Divisor

Beispiel:

```
INT iResult

iResult=mal_modulo(10, 3)
//iResult==1

iResult=mal_modulo(19, 4)
//iResult==3

iResult=mal_modulo(19, -4)
//iResult==3

iResult=mal_modulo(-19, 4)
//iResult==3

iResult=mal_modulo(-19, -4)
//iResult==3
```

6.1.5.2 mal_2pow (...)

Beschreibung:

Potenzfunktion mit ganzzahligen Exponenten zur Basis 2

Funktionskopf:

GLOBAL DEFFCT INT mal_2pow(exponent:IN)

Rückgabe:

Potenzwert

Parameter:

Parameter	Datentyp	Beschreibung
exponent	UINT	Exponent

Beispiel:

```
INT iResult

iResult=mal_2pow(3)
//iResult==8
```

6.1.5.3 *mal_powInt (...)*

Beschreibung:

Potenzfunktion mit ganzzahligen Exponenten und reeller Basis.

Funktionskopf:

GLOBAL DEFFCT REAL mal_powInt(base :IN, exponent :IN)

Rückgabe:

Potenzwert

Parameter:

Parameter	Datentyp	Beschreibung
base	REAL	Basis
exponent	INT	Exponent

Beispiel:

```
REAL result, base
INT exponent

base=15
exponent=2

result=mal_powInt(base, exponent)
//result==225

base=1.5
exponent=-5

result=mal_powInt(base, exponent)
// result==0.131687269
```

6.1.5.4 *mal_powUInt (...)*

Beschreibung:

Potenzfunktion mit vorzeichenlosen, ganzzahligen Exponenten und reeller Basis.

Funktionskopf:

GLOBAL DEFFCT REAL mal_powUInt(base :IN, uExponent :IN)

Rückgabe:

Potenzwert.

Parameter:

Parameter	Datentyp	Beschreibung
base	REAL	Basis

uExponent	INT	vorzeichenloser Exponent
-----------	-----	--------------------------

Beispiel:

```
REAL result, base
INT exponent

base=7.8
exponent=4

result=mal_powInt(base, exponent)
//result==3701.5056
```

6.1.5.5 mal_hypot (...)

Beschreibung:

Quadratwurzel der Summe zweier Quadrate. Satz des Pythagoras.

Funktionskopf:

GLOBAL DEFFCT REAL mal_hypot (x :IN, y:IN)

Rückgabe:

Hypothenuse eines rechtwinkligen Dreieckes mit den Seiten x und y.

Parameter:

Parameter	Datentyp	Beschreibung
x	REAL	Seitenlänge des 1. Quadrates
y	REAL	Seitenlänge des 2. Quadrates

Beispiel:

```
REAL x, y, result

x=3.0
y=4.0

result=mal_hypot(x, y)
//result==5.0
```

6.1.5.6 mal_hypot3(...)

Beschreibung:

Quadratwurzel der Summe dreier Quadrate.

Funktionskopf:

GLOBAL DEFFCT REAL mal_hypot3(x :IN, y:IN, z:IN)

Rückgabe:

Seitenlänge eines Quadrates, dass sich aus der Summer dreier Quadrate ergibt.

Parameter:

Parameter	Datentyp	Beschreibung
x	REAL	Seitenlänge des 1. Quadrates
y	REAL	Seitenlänge des 2. Quadrates
z	REAL	Seitenlänge des 3. Quadrates

Beispiel:

```
REAL x, y, z, result

x=3.0
y=4.0
z=5.0

result=mal_hypot3(x, y, z)
//result==7.07
```

6.1.6 Bit-Operationen

6.1.6.1 *mal_bitState(...)*

Beschreibung:

Zustandsprüfung eines Bits einer ganzen Zahl.

Funktionskopf:

GLOBAL DEFFCT BOOL mal_bitState(intValue:IN, bitNo:IN)

Rückgabe:

- True: Bit ist gesetzt.
- False: Bit ist nicht gesetzt.

Parameter:

Parameter	Datentyp	Beschreibung
intValue	INT	Ganzzahl zur Prüfung
bitNo	INT	Nummer des Bits, dessen Zustand geprüft wird (0...31)

Beispiel:

```
INT intValue, bitPattern[8], idx

intValue=55

FOR idx=0 TO 7
    bitPattern[idx+1]=mal_bitState(intValue, idx)
ENDFOR

// bitPattern[0]=TRUE
// bitPattern[1]=TRUE
// bitPattern[2]=TRUE
```

```
// bitPattern[3]=FALSE
// bitPattern[4]=TRUE
// bitPattern[5]=TRUE
// bitPattern[6]=FALSE
// bitPattern[7]=FALSE
```

6.1.6.2 *mal_shiftBitsRight(...)*

Beschreibung:

Logische Bit-Verschiebung einer Ganzzahl nach rechts. Niederwertige Bits werden verworfen.

Funktionskopf:

GLOBAL DEFECT INT mal_shiftBitsRight(intValue:IN, bitsNo:IN)

Rückgabe:

Ergebnis der Bit-Verschiebung.

Parameter:

Parameter	Datentyp	Beschreibung
intValue	INT	Ganzzahl, dessen Bits verschoben werden. (>=0)
bitsNo	INT	Anzahl der Bits, um die verschoben wird

Beispiel:

```
INT intValue, bitsNo, iResult

intValue=23
bitsNo=2

iResult=mal_shiftBitsRight(intValue, bitsNo)

// intValue='B10111'
// iResult=5='B0101'
```

6.1.7 Sortieralgorithmen

6.1.7.1 *mal_quickSort(...)*

Beschreibung:

Quicksort-Algorithmus. Sortierung der Elemente eines Vektors in aufsteigender Reihenfolge.

Funktionskopf:

GLOBAL DEF mal_qsort(vector[:OUT, n:IN)

Rückgabe:

Vektor, mit einer Sortierung der Elemente in aufsteigender Reihenfolge.

Parameter:

Parameter	Datentyp	Beschreibung
vector[]	REAL	Eindimensionaler Vektor
n	INT	Anzahl Elemente des Vektors

Beispiel:

```
REAL vector[8]
```

```
vector[1]=4
vector[2]=3
vector[3]=5
vector[4]=2
vector[5]=1
vector[6]=3
vector[7]=2
vector[8]=3
```

```
mal_quickSort(vector[], 8)
```

```
// vector[1]=1
// vector[2]=2
// vector[3]=2
// vector[4]=3
// vector[5]=3
// vector[6]=3
// vector[7]=4
// vector[8]=5
```

6.1.8 Vektoren

6.1.8.1 mal_vectorLength(...)

Beschreibung:

Betrag eines Vektors.

Funktionskopf:

```
GLOBAL DEFFCT REAL mal_vectorLength(v[:OUT, n:IN)
```

Rückgabe:

Betrag des Vektors.

Parameter:

Parameter	Datentyp	Beschreibung
vector[]	REAL	Eindimensionaler Vektor
n	INT	Anzahl Elemente des Vektors

Beispiel:

```
REAL rVector[2]
REAL vectorLength
```

```

rVector[1]=6.0
rVector[2]=8.0
vectorLength=mal_vectorLength(rVector[], 2)

// vectorLength=10.0

```

6.1.8.2 **mal_vectorProduct(...)**

Beschreibung:

Vektorprodukt zweier Vektoren im 3-dimensionalen Raum.

Funktionskopf:

GLOBAL DEF mal_vectorProduct(vector_a[:OUT, vector_b[:OUT, vectorProduct[:OUT)

Rückgabe:

Vektorprodukt.

Parameter:

Parameter	Datentyp	Beschreibung
vector_a[]	REAL	dreidimensionaler Vektor a
vector_b[]	REAL	dreidimensionaler Vektor b
vectorProduct[]	REAL	Vektorprodukt: vector_a[] x vector_b[]

Beispiel:

```

REAL vector_a[3], vector_b[3], vectorProduct[3]

vector_a[1]=4
vector_a[2]=3
vector_a[3]=0
vector_b[1]=0
vector_b[2]=-20
vector_b[3]=0
mal_vectorProduct(vector_a[], vector_b[], vectorProduct[])

// vectorProduct[1]=0
// vectorProduct[2]=0
// vectorProduct[3]=-80

```

6.1.8.3 **mal_tripleProduct(...)**

Beschreibung:

Spatprodukt dreier Vektoren im 3-dimensionalen Raum. $\vec{a} * (\vec{b} \times \vec{c})$

Funktionskopf:

GLOBAL DEFFCT REAL mal_tripleProduct(a[:OUT, b[:OUT, c[:OUT)

Rückgabe:

Spatprodukt.

Parameter:

Parameter	Datentyp	Beschreibung
vector_a[]	REAL	dreidimensionaler Vektor a
vector_b[]	REAL	dreidimensionaler Vektor b
vector_c[]	REAL	dreidimensionaler Vektor c

Beispiel:

```
REAL vector_a[3], vector_b[3], vector_c[3], tripleProduct

vector_a[1]=2
vector_a[2]=0
vector_a[3]=5

vector_b[1]=-1
vector_b[2]=5
vector_b[3]=-2

vector_c[1]=2
vector_c[2]=1
vector_c[3]=2

tripleProduct=mal_tripleProduct(vector_a[], vector_b[], vector_c[])

//tripleProduct=-31
```

6.1.8.4 *mal_vectorAngle(...)*

Beschreibung:

Winkel zwischen zwei Vektoren im 3-dimensionalen Raum.

Funktionskopf:

GLOBAL DEFFCT REAL mal_vectorAngle(vector_a[:OUT, vector_b[:OUT)

Rückgabe:

Winkel in Grad.

Parameter:

Parameter	Datentyp	Beschreibung
vector_a[]	REAL	Dreidimensionaler Vektor a
vector_b[]	REAL	Dreidimensionaler Vektor b

Beispiel:

```
REAL vector_a[3], vector_b[3], phi

vector_a[1]=3
vector_a[2]=-1
vector_a[3]=2
vector_b[1]=1
vector_b[2]=2
vector_b[3]=4
phi=mal_vectorAngle(vector_a[], vector_b[])

// phi == 58.34
```

6.1.8.5 *mal_vectorMinMax(...)*

Beschreibung:

Bestimmung des minimalen und maximalen Wertes eines Vektors.

Funktionskopf:

GLOBAL DEF mal_vectorMinMax(vector[]:OUT, n:IN, min_out:OUT, max_out:OUT)

Rückgabe:

Minimaler Wert eines Vektors.

Maximaler Wert eines Vektors.

Parameter:

Parameter	Datentyp	Beschreibung
vector[]	REAL	Eindimensionaler Vektor.
n	INT	Anzahl Elemente des Vektors.
min_out	REAL	Minimaler Wert des Vektors.
max_out	REAL	Maximaler Wert des Vektors.

Beispiel:

```
REAL minVal, maxVal
```

```
REAL rVector[10]
rVector[1]=54.98
rVector[2]=546.759
rVector[3]=345.33
rVector[4]=243.345
rVector[5]=123.56
rVector[6]=9.45
rVector[7]=786.456
rVector[8]=342.5435
rVector[9]=2343.234
rVector[10]=567.565
```

```
mal_vectorMinMax(rVector[], 10, minVal, maxVal)
```

```
// minVal== 9.45
// maxVal== 2343.234
```

6.1.8.6 *mal_vectorAverage(...)*

Beschreibung:

Bestimmung des Mittelwertes der Elementes eines Vektors.

Funktionskopf:

GLOBAL DEF mal_vectorAverage(vector[]:OUT, n:IN, average_out:OUT)

Rückgabe:

Mittelwert der Elemente eines Vektors.

Parameter:

Parameter	Datentyp	Beschreibung
vector[]	REAL	Eindimensionaler Vektor.
n	INT	Anzahl Elemente des Vektors.
average_out	REAL	Mittelwert der Elemente des Vektors.

Beispiel:

```
REAL average
REAL rVector[10]
rVector[1]=54.98
rVector[2]=546.759
rVector[3]=345.33
rVector[4]=243.345
rVector[5]=123.56
rVector[6]=9.45
rVector[7]=786.456
rVector[8]=342.5435
rVector[9]=2343.234
rVector[10]=567.565

mal_vectorAverage(rVector[], 10, average)

//average==536.322205
```

6.1.8.7 **mal_vectorAvgMinMax (...)**

Beschreibung:

Bestimmung des minimalen und maximalen Wertes eines Vektors sowie dessen Mittelwert.

Funktionskopf:

```
GLOBAL DEF mal_vectorAvgMinMax(vector[:OUT, n:IN, average_out:OUT, min_out:OUT,
max_out:OUT)
```

Rückgabe:

Minimaler Wert eines Vektors.

Maximaler Wert eines Vektors.

Mittelwert der Elemente eines Vektors.

Parameter:

Parameter	Datentyp	Beschreibung
vector[]	REAL	Eindimensionaler Vektor.
n	INT	Anzahl Elemente des Vektors.
average_out	REAL	Mittelwert der Elemente des Vektors.
min_out	REAL	Minimaler Wert des Vektors.
max_out	REAL	Maximaler Wert des Vektors.

Beispiel:

```
REAL average, minVal, maxVal
REAL rVector[10]
rVector[1]=54.98
rVector[2]=546.759
rVector[3]=345.33
rVector[4]=243.345
rVector[5]=123.56
rVector[6]=9.45
rVector[7]=786.456
rVector[8]=342.5435
rVector[9]=2343.234
rVector[10]=567.565

mal_vectorAvgMinMax(rVector[], 10, average, minVal, maxVal)

//average==536.322205
// minVal== 9.45
// maxVal== 2343.234
```

6.1.8.8 mal_vectorNormalization (...)

Beschreibung:

Berechnung eines Vektors der Länge Eins.

Funktionskopf:

GLOBAL DEF mal_vectorNormalization(vector[:OUT, n:IN)

Rückgabe:

Vektor der Länge Eins

Parameter:

Parameter	Datentyp	Beschreibung
vector[]	REAL	Eindimensionaler Vektor.
n	INT	Anzahl Elemente des Vektors.

Beispiel:

```
REAL rVector[3]
rVector[1]=3
rVector[2]=-1
rVector[3]=2

mal_vectorNormalization(rVector[], 3)

//rVector[1]==0.801783681
//rVector[2]==-0.267261237
//rVector[3]==0.534522474
```

6.1.9 Matrizen

6.1.9.1 *mal_rotMatToRPY*

Beschreibung:

Berechnung der RPY-Winkel aus einer Rotationsmatrix.

Funktionskopf:

GLOBAL DEFFCT val_CoordinateAngles_T mal_rotMatToRPY(rotMat[,]:OUT)

Rückgabe:

Winkel A,B,C.

Parameter:

Parameter	Datentyp	Beschreibung
rotMat	REAL[3,3]	Rotationsmatrix

Beispiel:

```
DECL E6POS baseOrigin, baseX, baseXY
DECL REAL pXdiff[3], pXYdiff[3], py[3], pz[3], rotMat[3,3]

baseX=baseOrigin
baseX.X=baseX.X+100
baseX.Y=baseX.Y+100
baseX.Z=baseX.Z+100
baseXY=baseOrigin
baseXY.Y=baseXY.Y+100

pXdiff[1]=baseX.X-baseOrigin.X
pXdiff[2]=baseX.Y-baseOrigin.Y
pXdiff[3]=baseX.Z-baseOrigin.Z
pXYdiff[1]=baseXY.X-baseOrigin.X
pXYdiff[2]=baseXY.Y-baseOrigin.Y
pXYdiff[3]=baseXY.Z-baseOrigin.Z

mal_vectorNormalization(pXdiff[], 3)
mal_vectorNormalization(pXYdiff[], 3)

mal_vectorProduct(pXdiff[], pXYdiff[], pz[])
mal_vectorNormalization(pz[], 3)
mal_vectorProduct(pz[], pXdiff[], py[])

FOR i=1 TO 3
    rotMat[i,1] = pXdiff[i]
    rotMat[i,2] = py[i]
    rotMat[i,3] = pz[i]
ENDFOR

coordAngles=mal_rotMatToRPY(rotMat[,])

// coordAngles.A==45.0
// coordAngles.B==0.0
// coordAngles.C==0.0
```

6.1.10 Geometrie

6.1.10.1 *mal_circumference(...)*

Beschreibung:

Berechnung des Kreisumfangs.

Funktionskopf:

GLOBAL DEFFCT REAL mal_circumference(radius:IN)

Rückgabe:

Kreisumfang.

Parameter:

Parameter	Datentyp	Beschreibung
radius	REAL	Kreisradius

Beispiel:

```
REAL circumference, radius  
  
radius=10.0  
  
circumference=mal_circumference(radius)  
  
//circumference=62.83
```

6.1.10.2 *mal_circularArc(...)*

Beschreibung:

Berechnung des Kreisbogens.

Funktionskopf:

GLOBAL DEFFCT REAL mal_circularArc(radius:IN, angle:IN)

Rückgabe:

Kreisbogen

Parameter:

Parameter	Datentyp	Beschreibung
radius	REAL	Kreisradius
angle	REAL	Winkel [deg]

Beispiel:

```
REAL circularArc, radius, angle  
  
radius=10.0  
angle=60.0  
  
circularArc=mal_circularArc(radius, angle)
```

```
//circularArc=10.47
```

6.1.11 Zufallszahlen

6.1.11.1 *mal_randLcg(...)*

Beschreibung:

Erzeugung einer Pseudozufallszahl. Linearer Kongruenzgenerator.

Funktionskopf:

GLOBAL DEFFCT REAL mal_randLcg(seed:IN)

Rückgabe:

Real-Wert im Bereich 0.0...1.0.

Parameter:

Parameter	Datentyp	Beschreibung
seed	INT	Startwert, z.B \$ROB_TIMER

Beispiel:

```
REAL rand  
  
rand=mal_randLcg(123456789)  
  
// rand==0.153686523
```

6.1.11.2 *mal_randLcgRange(...)*

Beschreibung:

Erzeugung einer Pseudozufallszahl in einem Bereich. Linearer Kongruenzgenerator.

Funktionskopf:

GLOBAL DEFFCT REAL mal_randLcgRange(seed:IN, lowerBoundary:IN, upperBoundary:IN)

Rückgabe:

Real-Wert im Bereich lowerBoundary... upperBoundary.

Parameter:

Parameter	Datentyp	Beschreibung
seed	INT	Startwert, z.B \$ROB_TIMER
lowerBoundary	REAL	Untere Bereichsgrenze der Zufallszahl
upperBoundary	REAL	Obere Bereichsgrenze der Zufallszahl

Beispiel:

```
INT seed
REAL lowerBoundary, upperBoundary, rand

seed= 123456789
lowerBoundary= 12345
upperBoundary= 987654

rand=mal_randLcgRange(seed, lowerBoundary, upperBoundary)

// rand==162236.844
```

6.2 PhysLib: Physikalische Funktionen

6.2.1 Temperatur

6.2.1.1 *phl_linThermalExpansion(...)*

Beschreibung:

Berechnung der linearen thermischen Ausdehnung eines Körpers.

Funktionskopf:

GLOBAL DEFECT REAL phl_linThermalExpansion(lteCoeff:IN, particularLength:IN, tempDiff:IN)

Rückgabe:

Ausdehnung in [m].

Parameter:

Parameter	Datentyp	Beschreibung
lteCoeff	REAL	Ausdehnungskoeffizient [K^{-1}]
particularLength h	REAL	Länge [m]
tempDiff	REAL	Temperaturdifferenz [K]

Beispiel:

```
REAL partLength, tempDiff, thermalExpansion

partLength=1.2 ; [m]
tempDiff=45.5 ; [K]

thermalExpansion=phl_linThermalExpansion(phl_lteCopper, partLength, tempDiff)

// thermalExpansion==0.0009282
```

6.2.1.2 *phl_CelsiusToKelvin(...)*

Beschreibung:

Umrechnung von Celsius zu Kelvin.

Funktionskopf:

GLOBAL DEFFCT REAL *phl_CelsiusToKelvin*(celsius:IN)

Rückgabe:

Temperatur in Kelvin.

Parameter:

Parameter	Datentyp	Beschreibung
celsius	REAL	Temperatur in Celsius

Beispiel:

```
REAL tempKelvin, tempCelsius

tempCelsius=22.0
tempKelvin=phl_CelsiusToKelvin(tempCelsius)

// tempKelvin==295.15
```

6.2.1.3 *phl_KelvinToCelsius(...)*

Beschreibung:

Umrechnung von Kelvin zu Celsius.

Funktionskopf:

GLOBAL DEFFCT REAL *phl_KelvinToCelsius*(kelvin:IN)

Rückgabe:

Temperatur in Celsius.

Parameter:

Parameter	Datentyp	Beschreibung
kelvin	REAL	Temperatur in Kelvin

Beispiel:

```
REAL tempKelvin, tempCelsius

tempKelvin=295.15
tempCelsius=phl_KelvinToCelsius(tempKelvin)

// tempCelsius==22.0
```

6.2.1.4 *phl_CelsiusToFahrenheit(...)*

Beschreibung:

Umrechnung von Celsius zu Fahrenheit.

Funktionskopf:

GLOBAL DEFFCT REAL *phl_CelsiusToFahrenheit*(celsius:IN)

Rückgabe:

Temperatur in Fahrenheit.

Parameter:

Parameter	Datentyp	Beschreibung
celsius	REAL	Temperatur in Celsius

Beispiel:

```
REAL tempFahrenheit, tempCelsius

tempCelsius=-16.5
tempFahrenheit=phl_CelsiusToFahrenheit(tempCelsius)

// tempFahrenheit==2.3
```

6.2.1.5 *phl_FahrenheitToCelsius(...)*

Beschreibung:

Umrechnung von Fahrenheit zu Celsius.

Funktionskopf:

GLOBAL DEFFCT REAL *phl_FahrenheitToCelsius*(fahrenheit:IN)

Rückgabe:

Temperatur in Celsius.

Parameter:

Parameter	Datentyp	Beschreibung
fahrenheit	REAL	Temperatur in Fahrenheit

Beispiel:

```
REAL tempFahrenheit, tempCelsius

tempFahrenheit=2.3
tempCelsius=phl_FahrenheitToCelsius(tempFahrenheit)

// tempCelsius==-16.5
```

6.2.1.6 *phl_KelvinToFahrenheit(...)*

Beschreibung:

Umrechnung von Kelvin zu Fahrenheit.

Funktionskopf:

GLOBAL DEFFCT REAL *phl_KelvinToFahrenheit*(kelvin:IN)

Rückgabe:

Temperatur in Fahrenheit.

Parameter:

Parameter	Datentyp	Beschreibung
kelvin	REAL	Temperatur in Kelvin

Beispiel:

```
REAL tempFahrenheit, tempKelvin

tempKelvin=315.45
tempFahrenheit=phl_KelvinToFahrenheit(tempKelvin)

// tempFahrenheit==108.14
```

6.2.1.7 *phl_FahrenheitToKelvin(...)*

Beschreibung:

Umrechnung von Fahrenheit zu Kelvin.

Funktionskopf:

GLOBAL DEFFCT REAL *phl_FahrenheitToKelvin*(fahrenheit:IN)

Rückgabe:

Temperatur in Kelvin.

Parameter:

Parameter	Datentyp	Beschreibung
fahrenheit	REAL	Temperatur in Fahrenheit

Beispiel:

```
REAL tempFahrenheit, tempKelvin

tempFahrenheit =108.14
tempKelvin=phl_FahrenheitToKelvin(tempFahrenheit)

// tempKelvin==315.45
```

6.2.2 Zeit

6.2.2.1 *phl_isLeapYear(...)*

Beschreibung:

Prüfung auf Schaltjahr.

Funktionskopf:

GLOBAL DEFFCT BOOL phl_isLeapYear(year:IN)

Rückgabe:

- True: Jahr ist Schaltjahr.
- False: Jahr ist kein Schaltjahr.

Parameter:

Parameter	Datentyp	Beschreibung
year	INT	Jahr zu Prüfung auf Schaltjahr

Beispiel:

```
Bool isLeapYear
isLeapYear=phl_isLeapYear(2000)
// isLeapYear==TRUE
isLeapYear=phl_isLeapYear(2001)
// isLeapYear==FALSE
```

6.2.2.2 *phl_daysInMonth(...)*

Beschreibung:

Berechnung der Anzahl Tage in einem Monat.

Funktionskopf:

GLOBAL DEFFCT INT phl_daysInMonth(month:IN, year:IN)

Rückgabe:

Anzahl Tage im Monat.

Parameter:

Parameter	Datentyp	Beschreibung
month	INT	Monat im Jahr [1...12]
year	INT	Jahreszahl

Beispiel:

```

INT daysInMonth

daysInMonth=phl_daysInMonth(2, 2000)

// daysInMonth==29

daysInMonth=phl_daysInMonth(3, 2001)

// daysInMonth==31

```

6.2.2.3 *phl_daysInYear(...)*

Beschreibung:

Berechnung der Anzahl Tage in einem Jahr.

Funktionskopf:

GLOBAL DEFFCT INT phl_daysInYear(year:IN)

Rückgabe:

Anzahl Tage im Jahr.

Parameter:

Parameter	Datentyp	Beschreibung
year	INT	Jahreszahl

Beispiel:

```

INT daysInYear

daysInYear=phl_daysInYear(2000)

// daysInYear==366

daysInYear=phl_daysInYear(2001)

// daysInYear==365

```

6.2.2.4 *phl_daysElapsedOfYear(...)*

Beschreibung:

Berechnung der Anzahl Tage in dem Jahr bis zu einem Datum.

Funktionskopf:

GLOBAL DEFFCT INT phl_daysElapsedOfYear(actDate:IN)

Rückgabe:

Anzahl Tage in dem Jahr bis zu einem Datum.

Parameter:

Parameter	Datentyp	Beschreibung
actDate	DATE	Datum

Beispiel:

```
DECL DATE myDate
INT elapsedDays

myDate={SEC 45,MIN 34,HOURL 12,DAY 10,MONTH 10,YEAR 2021}

elapsedDays=phl_daysElapsedOfYear(myDate)

// elapsedDays==282
```

6.2.2.5 *phl_secInYear(...)*

Beschreibung:

Berechnung der Sekunden in einem Jahr.

Funktionskopf:

GLOBAL DEFFCT INT phl_secInYear(year:IN)

Rückgabe:

Sekunden in einem Jahr.

Parameter:

Parameter	Datentyp	Beschreibung
year	INT	Jahreszahl

Beispiel:

```
INT secInYear

secInYear=phl_secInYear(2000)

// secInYear==31622400

secInYear=phl_secInYear(2001)

// secInYear==31536000
```

6.2.2.6 *phl_secElapsedOfYear(...)*

Beschreibung:

Berechnung der Anzahl Sekunden in dem Jahr bis zu einem Datum.

Funktionskopf:

GLOBAL DEFFCT INT phl_secElapsedOfYear(actDate:IN)

Rückgabe:

Anzahl Sekunden in dem Jahr bis zu einem Datum.

Parameter:

Parameter	Datentyp	Beschreibung
actDate	DATE	Datum

Beispiel:

```
DECL DATE myDate
INT elapsedSec

myDate={SEC 58,MIN 23,HOURL 8,DAY 12,MONTH 8,YEAR 2020}

elapsedSec=phl_secElapsedOfYear(myDate)

// elapsedSec==19383838
```

6.2.2.7 *phl_secSince2000(...)*

Beschreibung:

Berechnung der Sekunden seit dem Jahr 2000 und einem Datum.

Funktionskopf:

GLOBAL DEFFCT INT phl_secSince2000(actDate:IN)

Rückgabe:

Anzahl Sekunden.

Parameter:

Parameter	Datentyp	Beschreibung
actDate	DATE	Datum

Beispiel:

```
DECL DATE myDate
INT elapsedSec

myDate={SEC 16,MIN 16,HOURL 6,DAY 18,MONTH 8,YEAR 2016}

elapsedSec=phl_secSince2000(myDate)

// elapsedSec==524816176
```

6.2.2.8 *phl_timeSpan(...)*

Beschreibung:

Berechnet die Zeitspanne zwischen zwei Datums-Strukturen.

Funktionskopf:

GLOBAL DEFFCT DATE phl_timeSpan(date1in:IN, date2in:IN)

Rückgabe:

Zeitspanne als Datums-Struktur.

Parameter:

Parameter	Datentyp	Beschreibung
date1in	DATE	Datum1
date2in	DATE	Datum2

Beispiel:

```
DECL DATE date1, date2, dateSpan

date1={SEC 55,MIN 56,HOURL 15,DAY 10,MONTH 4,YEAR 2004}
date2={SEC 12,MIN 8,HOURL 10,DAY 11,MONTH 2,YEAR 2008}

dateSpan=phl_timeSpan(date1, date2)

// dateSpan=={SEC 17,MIN 11,HOURL 18,DAY 0,MONTH 10,YEAR 3}
```

6.2.2.9 *phl_timeSpanDays(...)*

Beschreibung:

Berechnung der Zeitspanne in Tagen zwischen zwei Datums-Strukturen.

Funktionskopf:

GLOBAL DEFFCT INT phl_timeSpanDays(date1in:IN, date2in:IN)

Rückgabe:

Anzahl Tage.

Parameter:

Parameter	Datentyp	Beschreibung
date1in	DATE	Datum1
date2in	DATE	Datum2

Beispiel:

```
DECL DATE date1, date2, dateSpanDays

date1={SEC 44,MIN 45,HOURL 4,DAY 12,MONTH 1,YEAR 2003}
date2={SEC 55,MIN 56,HOURL 15,DAY 10,MONTH 4,YEAR 2004}

dateSpanDays=phl_timeSpanDays(date1, date2)
// dateSpanDays==454

date1={SEC 10,MIN 0,HOURL 0,DAY 5,MONTH 1,YEAR 2002}
date2={SEC 10,MIN 0,HOURL 0,DAY 5,MONTH 1,YEAR 2000}
```

```
dateSpanDays=phl_timeSpanDays(date1, date2)
// dateSpanDays==731
```

6.2.2.10 *phl_dateCompare(...)*

Beschreibung:

Vergleich zweier DATE-Strukturen.

Funktionskopf:

GLOBAL DEFFCT INT phl_dateCompare(date1In:IN, date2In:IN)

Rückgabe:

- 1: date1In > date2In.
- 0: date1In == date2In.
- -1: date1In < date2In.

Parameter:

Parameter	Datentyp	Beschreibung
date1In	DATE	Datum
date2In	DATE	Datum

Beispiel:

```
DECL DATE date1, date2
INT dateCompare

date1={SEC 55,MIN 56,HOURL 15,DAY 10,MONTH 4,YEAR 2004}
date2={SEC 12,MIN 8,HOURL 10,DAY 11,MONTH 2,YEAR 2008}

dateCompare=phl_dateCompare(date1, date2)
// dateCompare==-1

date1={SEC 12,MIN 8,HOURL 10,DAY 11,MONTH 2,YEAR 2008}
date2={SEC 12,MIN 8,HOURL 10,DAY 11,MONTH 2,YEAR 2008}

dateCompare=phl_dateCompare(date1, date2)
// dateCompare==0

date1={SEC 12,MIN 8,HOURL 10,DAY 11,MONTH 2,YEAR 2008}
date2={SEC 55,MIN 56,HOURL 15,DAY 10,MONTH 4,YEAR 2004}

dateCompare=phl_dateCompare(date1, date2)
// dateCompare==1
```

6.2.2.11 *phl_dateStamp(...)*

Beschreibung:

Generiert aus einer DATE-Struktur einen Datumsstempel.

Funktionskopf:

GLOBAL DEFFCT INT phl_dateStamp(dateIn:IN)

Rückgabe:

Stempel des Datums in dem Format [yyyymmdd].

Parameter:

Parameter	Datentyp	Beschreibung
dateIn	DATE	Datum

Beispiel:

```
DECL DATE date1
INT dateStamp

date1={SEC 55,MIN 56, HOUR 15, DAY 10, MONTH 4, YEAR 2004}

dateStamp=phl_dateStamp(date1)

//dateStamp==20040410
```

6.2.2.12 *phl_timeStamp(...)*

Beschreibung:

Generiert aus einer DATE-Struktur einen Zeitstempel.

Funktionskopf:

GLOBAL DEFFCT INT phl_timeStamp(dateIn:IN)

Rückgabe:

Stempel der Zeit in dem Format [hhmmss].

Parameter:

Parameter	Datentyp	Beschreibung
dateIn	DATE	Datum

```
DECL DATE date1
INT timeStamp

date1={SEC 55,MIN 56, HOUR 15, DAY 10, MONTH 4, YEAR 2004}

timeStamp=phl_timeStamp(date1)

//timeStamp==155655
```

6.3 RobLib: Roboterspezifische Funktionen

6.3.1 Transformation

6.3.1.1 **rol_ForwardTransform(...)**

Beschreibung:

Berechnung der kartesischen Koordinaten auf Basis von Achswerten.

Funktionskopf:

GLOBAL DEFFCT E6POS rol_ForwardTransform(axisValues: IN, softEndCheck:IN, errState: OUT)

Rückgabe:

Kartesische Position als Ergebnis der Vorwärtstransformation.

Fehlerstatus:

- -4: \$TOOL ungültig.
- -3: \$BASE ungültig.
- -2: Fehlerstatus ungültig.
- -1: nicht alle Achswerte initialisiert.
- 0: Berechnung erfolgreich.
- 1: Verletzung des Software-Endschalters.
- 2: Fehler bei Berechnung Vorwärtstransformation

Parameter:

Parameter	Datentyp	Beschreibung
axisValues	E6AXIS	Achswerte
softEndCheck	BOOL	Prüfung auf Verletzung von Software-Endschalter
errState	INT	Fehlerstatus

Beispiel:

```
DECL E6POS forwardPos
INT errState

forwardPos=rol_ForwardTransform($AXIS_ACT, TRUE, errState)

// result depends on mechanic and kinematic chain
// forwardPos=={E6POS: X 1702.02795, Y -1.83519944E-07, Z 1631.97205, A -180, B 45,C 180,S 2,T 10,E1
36.9205704,E2 0, E3 0, E4 0, E5 0,E6 0 }

// errState==0; Calculation successful
```

6.3.1.2 **rol_BackwardTransform(...)**

Beschreibung:

Berechnung der Achswerte auf Basis von kartesischen Koordinaten.

Funktionskopf:

GLOBAL DEFFCT E6AXIS rol_BackwardTransform(cartPosition: IN, startAxis: IN, errState: OUT)

Rückgabe:

Achswerte als Ergebnis der Rückwärtstransformation.

Fehlerstatus:

- -4: \$TOOL ungültig.
- -3: \$BASE ungültig.
- -2: Fehlerstatus ungültig.
- -1: nicht alle Positionswerte initialisiert.
- 0: Berechnung erfolgreich.
- 1: Verletzung des Software-Endschalters.
- 2: Verletzung des Arbeitsraumes.
- 3: Achswerte berechnet, Verletzung Software-Endschalter durch Turn-Wert

Parameter:

Parameter	Datentyp	Beschreibung
cartPosition	E6POS	Kartesische Position
startAxis	E6AXIS	Startwert Status und Turn, falls cartPosition diese nicht enthält
errState	INT	Fehlerstatus

Beispiel:

```
DECL E6POS destPos
DECL E6AXIS backwardAxis
INT errState

destPos=={E6POS: X 1702.02795, Y -1.83519944E-07, Z 1631.97205, A -180, B 45,C 180,S 2,T 10,E1
36.9205704,E2 0, E3 0, E4 0, E5 0,E6 0 }

backwardAxis=rol_BackwardTransform(destPos, $AXIS_ACT, errState)

// result depends on mechanic and kinematic chain.
// backwardAxis=={E6AXIS: A1 6.78381973E-09, A2 -90, A3 90, A4 -6.7838366E-09,A5 45,A6
9.5937871E-09,E1 36.9205704,E2 0,E3 0,E4 0,E5 0,E6 0}

// errState==0; Calculation successful
```

6.3.1.3 **rol_BaseFromPositions(...)**

Beschreibung:

Berechnung von Basis Koordinaten anhand von drei Positionen.

Funktionskopf:

GLOBAL DEFFCT FRAME rol_BaseFromPositions(baseOrigin:IN, baseX:IN, baseXY:IN)

Rückgabe:

Frame als Basis-Koordinaten.

Parameter:

Parameter	Datentyp	Beschreibung
-----------	----------	--------------

baseOrigin	E6POS	Ursprung des Koordinatensystems
baseX	E6POS	Position auf der positiven X-Achse des Koordinatensystems
baseXY	E6POS	Position auf der positiven XY-Ebene des Koordinatensystems

Beispiel:

```
DECL E6POS baseOrigin, baseX, baseXY
DECL FRAME baseData

baseOrigin=$POS_ACT
baseX=baseOrigin
baseX.X=baseX.X+100
baseX.Y=baseX.Y+100
baseXY=baseOrigin
baseXY.Y=baseXY.Y+100

baseData=rol_BaseFromPositions(baseOrigin, baseX, baseXY)

// baseData.X==baseOrigin.X
// baseData.Y==baseOrigin.Y
// baseData.Z==baseOrigin.Z
// baseData.A==45.0
// baseData.B==0.0
// baseData.C==0.0
```

6.3.2 Testen auf Gleichheit

6.3.2.1 rol_FrameCompare(...)

Beschreibung:

Diese Funktion bestimmt ob zwei Frame – Strukturen innerhalb einer Distanz- und Winkelgrenze liegen.

Funktionskopf:

GLOBAL DEFFCT BOOL rol_FrameCompare(frame1:IN, frame2:IN, distLimit:IN, angleLimit:IN)

Rückgabe:

- True: Distanz und Winkel liegen innerhalb der Grenzwerte.
- False: Distanz oder Winkel überschreiten den Grenzwert

Parameter:

Parameter	Datentyp	Beschreibung
frame1	FRAME	Vergleichsframe1
frame2	FRAME	Vergleichsframe2
distLimit	REAL	Grenzwert für den Abstand der Frame-Strukturen
angleLimit	REAL	Grenzwert für den Winkel der Frame-Strukturen

Beispiel:

```
DECL FRAME frame1, frame2
DECL REAL distLimit, angleLimit
DECL BOOL bReturn
```

```

distLimit=10.0
angleLimit=1.0

frame1={X 100, Y 200, Z 300, A 0, B 0, C 0}
frame2={X 105, Y 205, Z 305, A 0, B 0, C 0}

bReturn=rol_FrameDistance(frame1, frame2, distLimit, angleLimit)
// bReturn==TRUE

```

6.3.3 Distanzen

6.3.3.1 rol_FrameDistance(...)

Beschreibung:

Distanz zwischen zwei FRAME-Strukturen.

Funktionskopf:

GLOBAL DEFFCT REAL rol_FrameDistance(frame1:IN, frame2:IN)

Rückgabe:

Distanzwert in [mm] zwischen frame1 und frame2.

Parameter:

Parameter	Datentyp	Beschreibung
frame1	FRAME	Vergleichsframe1
frame2	FRAME	Vergleichsframe2

Beispiel:

```

DECL FRAME frame1, frame2
REAL frameDist

frame1={X 100, Y 100, Z 100, A 0, B 0, C 0}
frame2={X 110, Y 120, Z 130, A 0, B 0, C 0}

frameDist=rol_FrameDistance(frame1, frame2)
// frameDist==37.41657387

```

6.3.3.2 rol_cartPointDistance(...)

Beschreibung:

Distanz zwischen zwei E6POS-Strukturen.

Funktionskopf:

GLOBAL DEFFCT REAL rol_cartPointDistance(pos1:IN, pos2:IN)

Rückgabe:

Distanzwert in [mm] zwischen pos1 und pos2.

Parameter:

Parameter	Datentyp	Beschreibung
pos1	E6POS	Vergleichsposition 1.
pos2	E6POS	Vergleichsposition 2.

Beispiel:

```
DECL E6POS pos1, pos2
REAL posDist

pos1={X 100.0, Y 100.0, Z 100.0, A 0.0, B 0.0, C 0.0, S 0, T 0, E1 0.0, E2 0.0, E3 0.0,E4 0.0,E5
0.0,E6 0.0}
pos2={X 110.0, Y 120.0, Z 130.0, A 0.0, B 0.0, C 0.0, S 0, T 0, E1 0.0, E2 0.0, E3 0.0,E4 0.0,E5
0.0,E6 0.0}

posDist=rol_cartPointDistance(pos1, pos2)
// posDist==37.41657387
```

6.3.4 Orientierungen

6.3.4.1 *rol_toolOriToGravity(...)*

Beschreibung:

Orientierung des Tool-Koordinatensystem zur Richtung der Gravitation.

Funktionskopf:

GLOBAL DEFFCT val_CoordinateAxes_T rol_toolOriToGravity(position:IN)

Rückgabe:

Winkelabweichungen in Grad der Koordinaten x,y und z zur Richtung der Gravitation.

Parameter:

Parameter	Datentyp	Beschreibung
position	E6POS	Position, gegen die die Orientierungsabweichung des Tool-Koordinatensystem berechnet wird.

Beispiel:

```
DECL val_CoordinateAxes_T coordAxes

$TOOL=$NULLFRAME
$BASE=$NULLFRAME
PTP {A1 0.0, A2 -90.0, A3 90.0, A4 0.0, A5 45.0, A6 0.0}

coordAxes=rol_toolOriToGravity($POS_ACT)

// coordAxes=={val_CoordinateAxes_T: x 45, y 90, z 45}
```

6.3.4.2 *rol_posDirToGravity(...)*

Beschreibung:

Ermittlung, ob die Bewegung zu einer Zielposition entgegen oder mit der Schwerkraft erfolgt.

Funktionskopf:

GLOBAL DEFFCT val_DirToGravity_T rol_posDirToGravity(startPos :IN, destPos :IN)

Rückgabe:

Enum:

- upGravity, Bewegung zur Zielposition erfolgt entgegen der Gravitation.
- downGravity, Bewegung zur Zielposition erfolgt mit der Gravitation.
- neutralGravity, Bewegung zur Zielposition erfolgt neutral der Gravitation.

Parameter:

Parameter	Datentyp	Beschreibung
startPos	E6POS	Startposition
destPos	E6POS	Zielposition

Beispiel:

```
DECL val_DirToGravity_T posDirToGrav
DECL E6POS startPos, destPos

$TOOL=$NULLFRAME
$BASE=$NULLFRAME

PTP {A1 0.0, A2 -90.0, A3 90.0, A4 0.0, A5 45.0, A6 0.0}
startPos=$POS_ACT
destPos=startPos
destPos.Z=startPos.Z + 20

posDirToGrav=rol_posDirToGravity(startPos, destPos)
//posDirToGrav==#upGravity

destPos.Z=startPos.Z
posDirToGrav=rol_posDirToGravity(startPos, destPos)
//posDirToGrav==#neutralGravity

destPos.Z=startPos.Z - 20
posDirToGrav=rol_posDirToGravity(startPos, destPos)
//posDirToGrav==#downGravity
```

6.3.5 Timerfunktion

6.3.5.1 rol_elapsedMsecRobTimer(...)

Beschreibung:

Zeitdifferenz zum aktuellen Wert des \$ROB_TIMER unter Berücksichtigung eines Überlaufes.

Funktionskopf:

GLOBAL DEFFCT INT rol_elapsedMsecRobTimer(compTime:IN)

Rückgabe:

Zeitdifferenz zwischen Parameter compTime und aktueller Zeit des \$ROB_TIMER in [ms].

Parameter:

Parameter	Datentyp	Beschreibung
compTime	INT	Vergleichswert für die Zeitdifferenz [ms]

Beispiel:

```
INT startTime, elapsedTime  
  
startTime=$ROB_TIMER  
WAIT SEC 1  
elapsedTime=rol_elapsedMsecRobTimer(startTime)  
  
//elapsedTime==1000
```

6.3.6 Roboter-Halt

6.3.6.1 rol_robStopAndRelease(...)

Beschreibung:

Stoppt die laufende Bewegung des Roboters und hebt den Stopp umgehend wieder auf. Die Fortsetzung der Bewegung erfordert einen Eingriff durch den Bediener. Die Funktion kann ausschließlich im Submitinterpreter verwendet werden.

Funktionskopf:

GLOBAL DEFFCT BOOL rol_robStopAndRelease(robStopType:IN)

Rückgabe:

- TRUE – Ein durch rol_robStop(...) eingeleiteter Stopp wurde aufgehoben.
- FALSE – Es wurde zuvor kein Stopp durch rol_robStop(...) eingeleitet.

Parameter:

Parameter	Datentyp	Beschreibung
robStopType	ROB_STOP_T	- #Ramp_Down (Rampenstopp). - #Path_Maintaining (bahntreuer Stopp).

6.3.6.2 rol_robStop(...)

Beschreibung:

Stoppt die laufende Bewegung des Roboters. Eine entsprechende Zustandsmeldung wird abgesetzt. Die Funktion kann ausschließlich im Submitinterpreter verwendet werden.

Funktionskopf:

GLOBAL DEFFCT BOOL rol_robStop(robStopType:IN)

Rückgabe:

- TRUE – Stopp wird erfolgreich ausgeführt.
- FALSE – Fehlerhafter Parameter.

Parameter:

Parameter	Datentyp	Beschreibung
robStopType	ROB_STOP_T	- #Ramp_Down (Rampenstopp). - #Path_Maintaining (bahntreuer Stopp).

6.3.6.3 **rol_robStopRelease(...)**

Beschreibung:

Ein durch rol_robStop(...) eingeleiteter Stopp wird aufgehoben. Die Funktion kann ausschließlich im Submitinterpreter verwendet werden.

Funktionskopf:

GLOBAL DEFFCT BOOL rol_robStopRelease()

Rückgabe:

- TRUE – Ein durch rol_robStop(...) eingeleiteter Stopp wurde aufgehoben.
- FALSE – Es wurde zuvor kein Stopp durch rol_robStop(...) eingeleitet.

Parameter:

keine Parameter

6.3.7 **Trace**

6.3.7.1 **rol_TraceStart(...)**

Beschreibung:

Starten einer Trace-Aufzeichnung.

Funktionskopf:

GLOBAL DEF rol_TraceStart(traceConfig:IN, traceName[:IN])

Parameter:

Parameter	Datentyp	Beschreibung
traceConfig[]	CHAR	Name der Trace-Konfiguration
traceName[]	CHAR	Name der Trace-Aufzeichnung. Ist der Parameter nicht initialisiert, wird als Aufzeichnungsname der Basisname aus der Konfiguration verwendet.

Beispiel:

```
rol_TraceStart("Tracedef_KRC_IpoCommandValues.xml", )  
// Systemmeldung: „Base name “KRC_IpoCommandValues” is being used for trace recording.”  
  
PTP_REL {X 100,Z -200}  
  
rol_TraceEnd()
```

6.3.7.2 *rol_TraceEnd()*

Beschreibung:

Stoppen einer laufenden Trace-Aufzeichnung.

Funktionskopf:

GLOBAL DEF rol_TraceEnd()

Parameter:

keine Parameter.

Beispiel:

siehe „rol_TraceStart(..)“

6.4 AxisLib: Achsspezifische Funktionen

6.4.1 Positionen

6.4.1.1 *axl_axisCmdPos(...)*

Beschreibung:

Kommandierte Position einer Achse.

Funktionskopf:

GLOBAL DEFFCT REAL axl_axisCmdPos(axNo:IN)

Rückgabe:

Aktuell kommandierte Position.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```
REAL cmdPos
INT axNo

axNo=3
cmdPos=axl_axisCmdPos(axNo)

// result depends on current commanded position
// cmdPos=84.0332
```

6.4.1.2 axl_axisMeasPos(...)

Beschreibung:

Istposition einer Achse.

Funktionskopf:

GLOBAL DEFFCT REAL axl_axisMeasPos(axNo:IN)

Rückgabe:

Aktuelle Istposition.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```
REAL actPos
INT axNo

axNo=7
actPos=axl_axisMeasPos(axNo)

// result depends on actual axis position
// actPos=142.0332
```

6.4.1.3 axl_axisMot(...)

Beschreibung:

Aktueller Motorwinkel.

Funktionskopf:

GLOBAL DEFFCT REAL axl_axisMot(axNo:IN)

Rückgabe:

Aktueller Motorwinkel in Grad.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```
REAL motPos
INT axNo

axNo=7
motPos=axl_axisMot(axNo)

// result depends on actual motor position
// motPos=1185
```

6.4.1.4 axl_axisPosInfo(...)

Beschreibung:

Informationen zu den Positionen einer Achse.

Funktionskopf:

GLOBAL DEFFCT axl_AxisPosInfo_T axl_axisPosInfo(axNo:IN)

Rückgabe:

Struktur vom Typ axl_AxisInfo_T mit folgenden Komponenten:

- posCmd: kommandierte Position [deg] / [mm].
- posAct: Istposition [deg] / [mm].
- posLag: Positionsschleppfehler [deg] / [mm].
- posMot: Motorwinkel der Achse [deg].

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```
DECL axl_AxisPosInfo_T axisPosInfo
INT axNo

axNo=1
axisPosInfo=axl_axisPosInfo(axNo)

// {axl_AxisPosInfo_T:
// posCmd 76.0269928,
// posAct 76.46698,
// posLag -0.439987183,
// posMot -16682.8184
// }
```

6.4.1.5 axl_exaxPosE6Pos(...)

Beschreibung:

Lesen der Position einer Zusatzachse in einer E6POS-Struktur.

Funktionskopf:

GLOBAL DEFFCT REAL axl_exaxPosE6Pos(exaxNo:IN, e6position:IN)

Rückgabe:

Position einer Zusatzachse.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)
e6position	E6POS	E6POS - Struktur

Beispiel:

```
REAL exaxPos
INT exaxNo
DECL E6POS destPos

exaxNo=2
destPos={E6POS: X 1702.0,Y 0.0,Z 1632.0,A -180.0,B 45.0,C -180.0,S 2,T 2,E1 120.0,E2 247.6103,E3
0,E4 0,E5 0,E6 0}

exaxPos=axl_exaxPosE6Pos(exaxNo, destPos)

// exaxPos=247.6103
```

6.4.1.6 axl_setExAxValueE6pos(...)

Beschreibung:

Setzen der Position einer Zusatzachse in einer E6POS-Struktur.

Funktionskopf:

GLOBAL DEFFCT E6POS axl_setExAxValueE6pos(exaxNo:IN, exaxPos:IN, posStrucIn:IN)

Rückgabe:

E6POS-Struktur mit Wert von exaxPos belegtem Strukturelement einer Zusatzachse.

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6).
exaxPos	REAL	Positionsvorgabe einer Zusatzachse.
posStrucIn	E6POS	E6POS-Struktur dessen Strukturelement für einer Zusatzachse belegt

		wird.
--	--	-------

Beispiel:

```
DECL E6POS oldPos, newPos
INT exaxNo
REAL exax2Pos

oldPos={E6POS: X 1702.0,Y 0.0,Z 1632.0,A -180.0,B 45.0,C -180.0,S 2,T 2,E1 120.0,E2 44.0,E3 0,E4
0,E5 0,E6 0}

exaxNo=2
exax2Pos=12.0
newPos=axl_setExAxValueE6pos(exaxNo, exax2Pos, oldPos)

// newPos={E6POS: X 1702.0, Y 0.0, Z 1632.0, A -180.0, B 45.0, C -180.0, S 2, T 2, E1 120.0, E2
12.0, E3 0, E4 0,E5 0, E6 0}
```

6.4.1.7 axl_setAxisValue(...)

Beschreibung:

Setzen der Position einer Achse in einer E6AXIS-Struktur.

Funktionskopf:

GLOBAL DEFFCT E6AXIS axl_setAxisValue(axNo:IN, axPos:IN, axisStrucIn:IN)

Rückgabe:

E6AXIS-Struktur mit Wert von axPos belegtem Strukturelement einer Achse.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12).
axPos	REAL	Positionsvorgabe einer Achse.
axisStrucIn	E6AXIS	E6AXIS-Struktur dessen Strukturelement für einer Zusatzachse belegt wird.

Beispiel:

```
DECL E6AXIS oldAxis, newAxis
INT axNo
REAL axPos

oldAxis={E6AXIS: A1 0.0, A2 -90.0, A3 90.0, A4 0.0, A5 45.0, A6 0.0, E1 0.0, E2 0.0, E3 0.0,E4
0.0,E5 0.0,E6 0.0}

axNo=8
axPos=12.0
newAxis=axl_setAxisValue(axNo, axPos, oldAxis)

// newAxis={E6AXIS: A1 0.0, A2 -90.0, A3 90.0, A4 0.0, A5 45.0, A6 0.0, E1 0.0, E2 12.0, E3 0.0,E4
0.0,E5 0.0,E6 0.0}
```

6.4.1.8 axl_getAxisValue(...)

Beschreibung:

Lesen der Position einer Achse in einer E6AXIS-Struktur.

Funktionskopf:

GLOBAL DEFFCT REAL axl_getAxisValue(axNo:IN, axisStrucIn:IN)

Rückgabe:

Position einer Achse.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12).
axisStrucIn	E6AXIS	E6AXIS-Struktur dessen Strukturelement für einer Zusatzachse belegt wird.

Beispiel:

```
DECL E6AXIS axisVariable
INT axNo
REAL axPos

axisVariable={E6AXIS: A1 0.0, A2 -90.0, A3 90.0, A4 0.0, A5 145.665, A6 0.0, E1 0.0, E2 0.0, E3
0.0,E4 0.0,E5 0.0,E6 0.0}

exaxNo=5
axPos=axl_getAxisValue(axNo, axisVariable)

// axPos= 145.665
```

6.4.1.9 axl_setE6axisElement(...)

Beschreibung:

Schreiben einer Achsposition in einer E6AXIS – Struktur.

Funktionskopf:

GLOBAL DEFFCT E6AXIS axl_setE6axisElement(axNo:IN, axPos:IN)

Rückgabe:

E6AXIS – Struktur mit Wert von axPos belegtem Strukturelement. Komponenten der anderen Achsen sind nicht initialisiert.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12).

axPos	REAL	Positionsvorgabe einer Achse.
-------	------	-------------------------------

Beispiel:

```
DECL E6AXIS axisRet
INT axNo
REAL axPos

axNo=3
axPos=123.456

axisRet=axl_setE6axisElement(axNo, axPos)

// axisRet={E6AXIS: A3 123.456}
```

6.4.2 Distanzen

6.4.2.1 axl_axisDistPosToPos(...)

Beschreibung:

Achsdifferenzen zweier E6POS-Strukturen.

Funktionskopf:

GLOBAL DEFFCT E6AXIS axl_axisDistPosToPos(startAxis: IN, pos1: IN, pos2:IN)

Rückgabe:

Achsdistanzen als E6AXIS-Struktur zwischen pos1 und pos2.

Parameter:

Parameter	Datentyp	Beschreibung
startAxis	E6AXIS	Startwert Status und Turn, falls pos1 diese nicht enthält.
pos1	E6POS	Vergleichsposition1
pos2	E6POS	Vergleichsposition2

Beispiel:

```
DECL E6POS pos1, pos2
DECL E6AXIS axisDist

pos1={E6POS: X 1599.38,Y -582.12,Z 1631.97,A 160,B 45,C -180,S 2,T 10,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}
pos2={E6POS: X 1599.38,Y 582.12,Z 1631.97,A -160,B 45,C 180,S 2,T 11,E1 0,E2 0,E3 0,E4 0,E5 0,E6 0}

axisDist = axl_axisDistPosToPos($AXIS_ACT, pos1, pos2)

// result depends on mechanic and kinematic chain.
// axisDist=={E6AXIS: A1 -40.0000038,A2 0,A3 0,A4 0,A5 0,A6 0,E1 0,E20,E3 0,E4 0,E5 0,E6 0}
```

6.4.2.2 axl_e6AxisDistToDestPos(...)

Beschreibung:

Differenzen der Achsen von der aktuellen Position zu einer Zielposition.

Funktionskopf:

GLOBAL DEFFCT E6AXIS axl_e6AxisDistToDestPos(destPos:IN)

Rückgabe:

Distanz aller Achsen.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
destPos	E6POS	Zielposition

Beispiel:

```
E6POS destPos
E6AXIS diffAxis

destPos=$POS_ACT
destPos.Z=destPos.Z + 20.0
diffAxis=axl_e6AxisDistToDestPos(destPos)

// The result depends on the mechanics.
// diffAxis==
// {E6AXIS:
// A1 -2.85059303E-16,
// A2 -0.0257339478,
// A3 -0.928970337,
// A4 3.13950475E-15,
// A5 0.954708099,
// A6 2.48747763E-15,
// E1 0,
// E2 0,
// E3 0,
// E4 0,
// E5 0,
// E6 0
// }
```

6.4.2.3 axl_exAxisDistToDestPos(...)

Beschreibung:

Differenz einer Zusatzachse von der aktuellen Position zu einer Zielposition.

Funktionskopf:

GLOBAL DEFFCT REAL axl_exAxisDistToDestPos(exaxNo:IN, destPos:IN)

Rückgabe:

Distanz einer Zusatzachse.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6).
destPos	E6POS	Zielposition.

Beispiel:

```
E6POS destPos
INT exaNo
REAL exaxDist

destPos=$POS_ACT
destPos.E1=destPos.E1 + 20.0
exaxNo=1

exaxDist=axl_exAxisDistToDestPos(exaxNo, destPos)

// exaxDist=20.0
```

6.4.2.4 axl_e6AxisDist(...)

Beschreibung:

Achsdifferenzen zweier E6AXIS-Strukturen.

Funktionskopf:

GLOBAL DEFFCT E6AXIS axl_e6AxisDist(startAxis:IN, destAxis:IN)

Rückgabe:

Differenz der Achsen zweier E6AXIS-Strukturen.

Einheit [mm] für eine Linearachse.

Einheit [deg] für eine rotatorische Achse.

Parameter:

Parameter	Datentyp	Beschreibung
startAxis	E6AXIS	Achsstellung1
destAxis	E6AXIS	Achsstellung2

Beispiel:

```
DECL E6AXIS startAxis, destAxis, diffAxis

startAxis={A1 10.0, A2 -70.0, A3 60.0, A4 0.0, A5 30.0, A6 -40.0, E1 120.0, E2 0.0, E3 0.0,E4 0.0,E5 0.0,E6 0.0}
destAxis={A1 -20.0, A2 -80.0, A3 70.0, A4 0.0, A5 20.0, A6 30.0, E1 240.0, E2 0.0, E3 0.0,E4 0.0,E5 0.0,E6 0.0}

diffAxis=axl_e6AxisDist(startAxis, destAxis)
```

```
// diffAxis={A1 -30.0, A2 -10.0, A3 10.0, A4 0.0, A5 -10.0, A6 70.0, E1 120.0, E2 0.0, E3 0.0,E4
0.0,E5 0.0,E6 0.0}
```

6.4.2.5 **axl_axisDist(...)**

Beschreibung:

Achsdifferenzen der Grundachsen zweier E6AXIS-Strukturen.

Funktionskopf:

GLOBAL DEFFCT AXIS axl_axisDist(startAxis:IN, destAxis:IN)

Rückgabe:

Differenz der Grundachsen zweier AXIS-Strukturen.

Parameter:

Parameter	Datentyp	Beschreibung
startAxis	AXIS	Grundachsstellung1
destAxis	AXIS	Grundachsstellung2

Beispiel:

```
DECL AXIS startAxis, destAxis, diffAxis

startAxis={A1 10.0, A2 -70.0, A3 60.0, A4 0.0, A5 30.0, A6 -40.0}
destAxis={A1 -20.0, A2 -80.0, A3 70.0, A4 0.0, A5 20.0, A6 30.0}

diffAxis=axl_axisDist(startAxis, destAxis)

// diffAxis={A1 -30.0, A2 -10.0, A3 10.0, A4 0.0, A5 -10.0, A6 70.0}
```

6.4.3 **Geschwindigkeiten**

6.4.3.1 **axl_maxVelOutput(...)**

Beschreibung:

Maximale abtriebsseitige Geschwindigkeit einer Achse.

Funktionskopf:

GLOBAL DEFFCT REAL axl_maxVelOutput(axNo:IN)

Rückgabe:

Abtriebsseitige Maximalgeschwindigkeit einer Achse.

Einheit [deg/s] für eine rotatorische Achse.

Einheit [mm/s] für eine Linearachse.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```

REAL outputSpeed
INT axNo

axNo=3
outputSpeed=axl_maxVelOutput(axNo)

// result depends on motor and gear properties as well as type of axis
// outputSpeed=112.0332

```

6.4.3.2 axl_absVelToRelVel(...)

Beschreibung:

Umrechnung einer Absolutgeschwindigkeit in eine relative Geschwindigkeit.

Funktionskopf:

GLOBAL DEFFCT REAL axl_absVelToRelVel(axNo:IN, absVel:IN)

Rückgabe:

Relativgeschwindigkeit (1..100%) einer Achse in Bezug zur Maximalgeschwindigkeit.

Einheit [%]

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)
absVel	REAL	Absolutgeschwindigkeit [mm/s] / [deg/s]

Beispiel:

```

REAL relSpeed, absSpeed
INT axNo

axNo=3
absSpeed=66.0; [deg/s]
relSpeed=axl_absVelToRelVel(axNo, absSpeed)

// result depends on max. velocity of axis
// relSpeed= 58.9111099; [%]

```

6.4.3.3 axl_relVelToAbsVel(...)

Beschreibung:

Umrechnung einer relativen Geschwindigkeit in einen absoluten Wert.

Funktionskopf:

GLOBAL DEFFCT REAL axl_relVelToAbsVel(axNo:IN, relVel:IN)

Rückgabe:

Abtriebsseitige Absolutgeschwindigkeit einer Achse.

Einheit [deg/s] für eine rotatorische Achse.

Einheit [mm/s] für eine Linearachse.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)
relVel	REAL	Relativgeschwindigkeit einer Achse in [%]

Beispiel:

```
REAL relSpeed, absSpeed
INT axNo

axNo=1
relSpeed=60.0;[%]
absSpeed=axl_relVelToAbsVel(axNo, relSpeed)

// result depends on max. velocity of axis
// absSpeed= 73.96212 ;[deg/s]
```

6.4.3.4 axl_axisVelInfo(...)

Beschreibung:

Informationen zu den Geschwindigkeiten einer Achse.

Funktionskopf:

GLOBAL DEFFCT axl_AxisVelInfo_T axl_axisVelInfo(axNo:IN)

Rückgabe:

Struktur vom Typ axl_AxisVelInfo_T mit folgenden Komponenten:

- velCmd: kommandierte abtriebsseitige Geschwindigkeit [deg/s] / [mm/s].
- velAct: abtriebsseitige Istgeschwindigkeit [deg/s] / [mm/s].
- velLag: abtriebsseitiger Positionsschleppfehler [deg/s] / [mm/s].

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12).

Beispiel:

```
DECL axl_AxisVelInfo_T axisVelInfo
INT axNo

axNo=1
axisVelInfo=axl_axisVelInfo(axNo)
```

```
// {axl_AxisVelInfo_T:
// velCmd 123.270187,
// velAct 79.6786346,
// velLag 43.5915527
// }
```

6.4.4 Momente

6.4.4.1 axl_axisTorqueInfo(...)

Beschreibung:

Informationen zu den Momenten einer Achse.

Funktionskopf:

GLOBAL DEFFCT axl_AxisTorqueInfo_T axl_axisTorqueInfo(axNo:IN)

Rückgabe:

Struktur vom Typ axl_TorqueInfo_T mit folgenden Komponenten:

- tqAct: Aktuelles Motormoment in [Nm] / [N].
- tqMax0: Maximales Dauer-Motormoment in [Nm] / [N].
- tqMax: Absolut maximales Motormoment in [Nm] / [N].
- tqHold: Haltemoment bei aktueller Achsstellung in [Nm] / [N].

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```
DECL axl_AxisTorqueInfo_T axisTorqueInfo
INT axNo
```

```
axNo=3
axisTorqueInfo=axl_axisTorqueInfo(axNo)
```

```
{axl_AxisTorqueInfo_T:
tqAct -3781.55298,
tqMax0 6507,
tqMax 17814.7207,
tqHold -3781.55298
}
```

6.4.5 Übersetzung

6.4.5.1 axl_linRatioSpindleDrive(...)

Beschreibung:

Umrechnung der Steigung eines Spindelgetriebes in das Übersetzungsverhältnis für Linearachsen.

Funktionskopf:

GLOBAL DEFFCT REAL axl_linRatioSpindleDrive(axNo:IN, spindlePitch:IN)

Rückgabe:

Linearübersetzung.

Einheit: [U/mm].

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)
spindlePitch	REAL	Spindelsteigung [mm/U]

Beispiel:

```
DECL axl_AxisTorqueInfo_T axisTorqueInfo
INT axNo
REAL spindlePitch, gearRatio

axNo=8
spindlePitch=5 ; [mm/U]
gearRatio=axl_linRatioSpindleDrive(axNo, spindlePitch)

// result depends gear ratio in $machine.dat
gearRatio=984.25
```

6.4.6 Achsenzustand

6.4.6.1 axl_axisMastered(...)

Beschreibung:

Prüfung ob ein gültige Justage für eine Achse vorliegt.

Funktionskopf:

GLOBAL DEFFCT BOOL axl_axisMastered(axNo:IN)

Rückgabe:

Zustand der Achsjustage:

- True: Achse justiert.
- False: Achse nicht justiert.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```

BOOL isMastered
INT axNo

axNo=1
isMastered=axl_axisMastered(axNo)

// isMastered==TRUE ;axis 1 is mastered
// isMastered==FALSE ;axis 1 is not mastered

```

6.4.6.2 axl_brakeOpen(...)

Beschreibung:

Prüfung ob Motorbremse einer Achse geöffnet ist.

Funktionskopf:

GLOBAL DEFFCT BOOL axl_brakeOpen(axNo:IN)

Rückgabe:

Zustand der Bremse einer Achse.

- True: Bremse geöffnet.
- False: Bremse geschlossen.

Parameter:

Parameter	Datentyp	Beschreibung
axNo	INT	Nummer der Achse (1...12)

Beispiel:

```

BOOL brakeOpen
INT axNo

axNo=1
brakeOpen=axl_brakeOpen(axNo)

// brakeOpen==TRUE ; brake axis 1 is open
// brakeOpen==FALSE ; brake axis 1 is closed

```

6.4.6.3 axl_exAxlAsynchron(...)

Beschreibung:

Prüfung ob sich eine Zusatzachse im asynchronen Betriebszustand befindet.

Funktionskopf:

GLOBAL DEFFCT BOOL axl_exAxlAsynchron(exaxNo :IN)

Rückgabe:

Synchroner / Asynchroner Betriebszustand der Zusatzachse:

- True: Achse asynchron.

- False: Achse synchron.

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)

Beispiel:

```

BOOL isAsynchron
INT exaxNo

exaxNo=3
isAsynchron=axl_exAxIsAsynchron(exaxNo)

// isAsynchron==TRUE ; external axis is asynchron
// isAsynchron==FALSE ; external axis is synchron

```

6.4.6.4 axl_exAxSynchron(...)

Beschreibung:

Zusatzachse in den Synchronbetrieb schalten.

Funktionskopf:

GLOBAL DEF axl_exAxSynchron(exaxNo:IN)

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)

Beispiel:

```

INT exaxNo

exaxNo=2
axl_exAxSynchron(exaxNo)

```

6.4.6.5 axl_exAxAsynchron(...)

Beschreibung:

Zusatzachse in den Asynchronbetrieb schalten.

Funktionskopf:

GLOBAL DEF axl_exAxAsynchron(exaxNo:IN)

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)

Beispiel:

```
INT exaxNo

exaxNo=2
axl_exAxAsynchron(exaxNo)

// Message: „E2 asynchronous external axis“
```

6.4.6.6 axl_exAxIsCoupled(...)

Beschreibung:

Prüfung ob eine Zusatzachse angekoppelt ist.

Funktionskopf:

GLOBAL DEFFCT BOOL axl_exAxIsCoupled(exaxNo :IN)

Rückgabe:

Zustand der Kopplung einer Zusatzachse:

- True: Achse gekoppelt
- False: Achse abgekoppelt

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)

Beispiel:

```
BOOL isCoupled
INT exaxNo

exaxNo=1
isCoupled=axl_exAxisCoupled(exaxNo)

// isCoupled==TRUE ; external axis is coupled
// isCoupled==FALSE ; external axis is decoupled
```

6.4.6.7 axl_exAxCouple(...)

Beschreibung:

Zusatzachse ankoppeln.

Das An- bzw. Abkoppeln erfordert eine entsprechende Konfiguration.

Datei: C:\KRC\Roboter\Config\User\Common\Mada\KRCAxes.xml

Attribut: Coupling

Funktionskopf:

GLOBAL DEF axl_exAxCouple(exaxNo:IN)

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)

Beispiel:

```
INT exaxNo  
  
exaxNo=2  
axl_exAxCouple(exaxNo)
```

6.4.6.8 axl_exAxDecouple(...)

Beschreibung:

Zusatzachse abkoppeln.

Das An- bzw. Abkoppeln erfordert eine entsprechende Konfiguration.

Datei: C:\KRC\Roboter\Config\User\Common\Mada\KRCAxes.xml

Attribut: Coupling

Funktionskopf:

GLOBAL DEF axl_exAxDecouple(exaxNo:IN)

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)

Beispiel:

```
INT exaxNo  
  
exaxNo=2  
axl_exAxDecouple(exaxNo)  
// Message: „E2 decoupled external axis“
```

6.4.7 Bewegungen

6.4.7.1 axl_exAxAsyncMovement(...)

Beschreibung:

Asynchroner Bewegungsbefehl für eine externe Achse.

Funktionskopf:

GLOBAL DEF axl_exAxAsyncMovement(exaxNo:IN, destPos:IN)

Parameter:

Parameter	Datentyp	Beschreibung
exaxNo	INT	Nummer der Zusatzachse (1...6)
destPos	REAL	Zielposition

Beispiel:

```
INT exaxNo  
REAL destPos
```

```
exaxNo=2  
destPos=120.0
```

```
axl_exAxAsyncMovement(exaxNo, destPos)
```

6.5 MsgLib: Anwendermeldungen

6.5.1 Meldungen abfeuern

6.5.1.1 *msl_FireMsg()*

Beschreibung:

Absetzen einer Meldung mit der Möglichkeit integer- oder real – Parameter zu übergeben. Die Summe der Parametern ist auf drei beschränkt.

Funktionskopf:

```
GLOBAL DEF msl_FireMsg(msgTyp:IN, msgModule[]):IN, msgNo:IN, msgTxt[]):IN, krlMsgOpt:IN,  
msgHandle:OUT, iPar1:IN, iPar2:IN, iPar3:IN, rPar4:IN, rPar5:IN, rPar6:IN)
```

Parameter:

Parameter	Datentyp	Beschreibung
msgTyp	Enum	Meldungstyp • #notify: Hinweismeldung. • #state: Zustandsmeldung. • #quit: Quittiermeldung. • #waiting: Wartemeldung.
msgModule[]	CHAR	Absender bzw. Name der kxr- Meldungsdatenbank • max. Zeichenlänge 26.
msgNo	INT	Meldungsnummer.
msgTxt[]	CHAR	Meldungstext: • max. Zeichenlänge: 80

krlMsgOpt	Struktur	Meldungsverhalten: Struktur vom Typ KrlMsgOpt_T • vl_stop ◦ TRUE: Meldungsausgabe löst Vorlaufstopp aus. ◦ FALSE: Kein Vorlaufstopp bei Meldungsausgabe • clear_p_reset ◦ TRUE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes gelöscht ◦ FALSE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes nicht gelöscht • clear_p_SAW ◦ TRUE: Meldungen werden bei Satzanwahl gelöscht ◦ FALSE: Meldungen werden bei Satzanwahl nicht gelöscht. • log_to_DB ◦ TRUE: Meldungen werden im Ereignisprotokoll geloggt ◦ FALSE: Meldungen werden nicht geloggt.
msgHandle	INT	Rückgabewert • -1: Die Meldung konnte nicht abgesetzt werden. • 0: Hinweismeldung. • >0: Die Meldung wurde erfolgreich abgesetzt.
iPar[1..3]	INT	Integer-Parameter 1...3.
rPar[1..3]	REAL	Real-Parameter 1...3.

Beispiel:

;Hinweismeldung

```
DECL KrlMsgOpt_T krlMsgOpt
INT msgHandle
```

```
krlMsgOpt.VL_Stop=TRUE
krlMsgOpt.Clear_P_Reset=TRUE
krlMsgOpt.Log_To_DB=FALSE
```

```
msl_FireMsg(#NOTIFY, „myKxrDataBase“, 12345, "squareRoot", krlMsgOpt, msgHandle, 3,,,SQRT(3),,)
```

```
; myKxrDataBase
```

```
<uiText key="squareRoot">
  <text xml:lang="de-DE">Die Quadratwurzel von %1 ist %2</text>
  <text xml:lang="en-US">The square root of %1 is %2</text>
  <text xml:lang="es-ES">La raíz cuadrada de %1 es %2</text>
</uiText>
```

```
//Message: „The square root of 3 is 1.732051“
// msgHandle==0
```

;Wartemeldung

```
$CYCFLAG[val_cy_msgLibWait]=$OUT[50];
; waiting message will cleared, when $CYCFLAG[val_cy_msgLibWait]=TRUE
```

```

msl_FireMsg(#WAITING, „myKxrDataBase“, 12348, "waitingOut", krlMsgOpt, msgHandle, 50,,,,,)

; myKxrDataBase

<uiText key="waitingOut">
  <text xml:lang="de-DE">Warten auf den Ausgang: %1</text>
  <text xml:lang="en-US">Waiting for the output: %1</text>
  <text xml:lang="es-ES">Esperando la salida: %1</text>
</uiText>

```

6.5.1.2 msl_FireDialog()

Beschreibung:

Absetzen einer Dialogmeldung mit der Möglichkeit integer oder real – Parameter zu übergeben. Die Anzahl von Parametern ist dabei auf drei beschränkt.

Funktionskopf:

GLOBAL DEF msl_FireDialog(msgModule[:IN, msgNo:IN, msgTxt[:IN, krlMsgOpt:IN, msgHandle:OUT, keySelected:OUT, keyTxt1[:IN, keyTxt2[:IN, keyTxt3[:IN, keyTxt4[:IN, keyTxt5[:IN, keyTxt6[:IN, keyTxt7[:IN, iPar1:IN, iPar2:IN, iPar3:IN, rPar4:IN, rPar5:IN, rPar6:IN)

Parameter:

Parameter	Datentyp	Beschreibung
msgModule[]	CHAR	Absender bzw. Name der kxr- Meldungsdatenbank: max. Zeichenlänge 26
msgNo	INT	Meldungsnummer.
msgTxt[]	CHAR	Meldungstext: max. Zeichenlänge: 80
krlMsgOpt	Struktur	Meldungsverhalten Struktur vom Typ KrlMsgOpt_T: - vl_stop - TRUE: Meldungsabgabe löst Vorlaufstopp aus. - FALSE: Kein Vorlaufstopp bei Meldungsabgabe. - clear_p_reset - TRUE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes gelöscht. - FALSE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes nicht gelöscht. - clear_p_SAW - TRUE: Meldungen werden bei Satzanwahl gelöscht. - FALSE: Meldungen werden bei Satzanwahl nicht gelöscht. - log_to_DB - TRUE: Logging der Meldungen im Ereignisprotokoll. - FALSE: kein Logging der Meldungen.

msgHandle	INT	Rückgabewert: • -1: Die Meldung konnte nicht abgesetzt werden. • >0: Die Meldung wurde erfolgreich abgesetzt.
keySelected	INT	Rückgabewert des vom Anwender selektierten softkeys.
KeyTxt[1...7]	CHAR	Bezeichner der Softkeys: • max. Zeichenlänge: 10.
IPar[1..3]	INT	Integer-Parameter.
RPar[1..3]	REAL	Real-Parameter.

Beispiel:

```
INT msgHandle, keySelected

msl_FireDialog(„myKxrDataBase“, 12346, "selectGripper", val_st_errMsgOpt, msgHandle, keySelected,
"1","2","3","4","5","6","7", , , , , )

; keySelect: returns number of selected key.

; myKxrDataBase

<uiText key="selectGripper">
  <text xml:lang="de-DE">Selektieren Sie den Greifer:</text>
  <text xml:lang="en-US">Select the gripper:</text>
  <text xml:lang="es-ES">Seleccione la pinza:</text>
</uiText>
```

6.5.1.3 msl_FireDialogQuitExt()

Beschreibung:

Absetzen einer Dialogmeldung mit der Möglichkeit mit der Möglichkeit einer Quittierung über Eingänge. Es kann jedem Softkey ein Eingang zugeordnet werden.

Dem Meldungstext können integer oder real – Parameter zu übergeben werden. Die Anzahl von Parametern ist dabei auf drei beschränkt.

Funktionskopf:

```
GLOBAL DEF msl_FireDialogQuitExt(msgModule[:IN, msgNo:IN, msgTxt[:IN, krlMsgOpt:IN,
msgHandle:OUT, keySelected:OUT, keyTxt1[:IN, keyTxt2[:IN, keyTxt3[:IN, keyTxt4[:IN,
keyTxt5[:IN, keyTxt6[:IN, keyTxt7[:IN, iPar1:IN, iPar2:IN, iPar3:IN, rPar4:IN, rPar5:IN, rPar6:IN,
di_QuitExt[:OUT)
```

Parameter:

Parameter	Datentyp	Beschreibung
msgModule[]	CHAR	Absender bzw. Name der kxr- Meldungsdatenbank: • max. Zeichenlänge 26.
msgNo	INT	Meldungsnummer.

msgTxt[]	CHAR	Meldungstext: max. Zeichenlänge: 80.
krlMsgOpt	Struktur	Meldungsverhalten. Struktur vom Typ KrlMsgOpt_T: - vl_stop - TRUE: Meldungsabgabe löst Vorlaufstopp aus. - FALSE: Kein Vorlaufstopp bei Meldungsabgabe. - clear_p_reset - TRUE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes gelöscht. - FALSE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes nicht gelöscht. - clear_p_SAW - TRUE: Meldungen werden bei Satzanwahl gelöscht. - FALSE: Meldungen werden bei Satzanwahl nicht gelöscht. - log_to_DB - TRUE: Meldungen werden im Ereignisprotokoll geloggt. - FALSE: Meldungen werden nicht geloggt.
msgHandle	INT	Rückgabewert -1: Die Meldung konnte nicht abgesetzt werden. >0: Die Meldung wurde erfolgreich abgesetzt.
keySelected	INT	Rückgabewert des vom Anwender selektierten softkeys oder des entsprechenden Eingangs.
keyTxt[1...7]	CHAR	Bezeichner der Softkeys: max. Zeichenlänge: 10
iPar[1..3]	INT	Integer-Parameter.
rPar[1..3]	REAL	Real-Parameter.
di_QuitExt[]	INT	Eingänge für die Dialogquittierung.

Beispiel:

```
DECL KrlMsgOpt_T krlMsgOpt
INT msgHandle, keySelected, di_QuitExt[7]

krlMsgOpt.VL_Stop=TRUE
krlMsgOpt.Clear_P_Reset=TRUE
krlMsgOpt.Log_To_DB=FALSE

di_QuitExt[1]=2056      ;Input for external acknowledgment: softkey „keyYes“
di_QuitExt[2]=2057      ;Input for external acknowledgment: softkey „keyNo“

msl_FireDialogQuitExt(„myKxrDataBase“, 12345, "forceExceeded", krlMsgOpt, msgHandle, keySelected,
"keyYes","keyNo",,,,, , 3, , , 4500, , ,di_QuitExt[])

; keySelect: returns number of selected key.
; keyYes→ 1
; keyNo → 2

; myKxrDataBase
```

```

<uiText key="forceExceeded">
  <text xml:lang="de-DE">Zange %1 - Überschreitung max. Kraft %2[N] Fortsetzen?</text>
  <text xml:lang="en-US">Gun %1 - exceeded force limit %2[N] Continue?</text>
  <text xml:lang="es-ES">Pinza %1 - excedido máx. forzar %2[N] Continuar?</text>
</uiText>
<uiText key="keyYes">
  <text xml:lang="de-DE">JA</text>
  <text xml:lang="en-US">YES</text>
  <text xml:lang="es-ES">SI</text>
</uiText>
<uiText key="keyNo">
  <text xml:lang="de-DE">NEIN</text>
  <text xml:lang="en-US">NO</text>
  <text xml:lang="es-ES">NO</text>
</uiText>

```

6.5.1.4 msl_FireStringParams()

Beschreibung:

Absetzen einer Meldung mit der Möglichkeit Zeichenketten als Parameter zu übergeben.

Funktionskopf:

GLOBAL DEF msl_FireStringParams(msgTyp:IN, msgModule[:IN, msgNo:IN, msgTxt[:IN, krlMsgOpt:IN, msgHandle:OUT, strPar1[:IN, strPar2[:IN, strPar3[:IN)

Parameter:

Parameter	Datentyp	Beschreibung
msgTyp	Enum	Meldungstyp: • #notify: Hinweismeldung. • #state: Zustandsmeldung. • #quit: Quittiermeldung. • #waiting: Wartemeldung.
msgModule[]	CHAR	Absender bzw. Name der kxr- Meldungsdatenbank: • max. Zeichenlänge 26.
msgNo	INT	Meldungsnummer.
msgTxt[]	CHAR	Meldungstext: • max. Zeichenlänge: 80.
krlMsgOpt	Struktur	Meldungsverhalten. Struktur vom Typ KrlMsgOpt_T: • vl_stop ◦ TRUE: Meldungsabgabe löst Vorlaufstopp aus. ◦ FALSE: Kein Vorlaufstopp bei Meldungsabgabe. • clear_p_reset ◦ TRUE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes gelöscht. ◦ FALSE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes nicht gelöscht. • clear_p_SAW

		◦ TRUE: Meldungen werden bei Satzanwahl gelöscht. ◦ FALSE: Meldungen werden bei Satzanwahl nicht gelöscht. • log_to_DB ◦ TRUE: Meldungen werden im Ereignisprotokoll geloggt. ◦ FALSE: Meldungen werden nicht geloggt.
msgHandle	INT	Rückgabewert: • -1: Die Meldung konnte nicht abgesetzt werden. • 0: Hinweismeldung. • >0: Die Meldung wurde erfolgreich abgesetzt.
strPar[1..3]	CHAR	String-Parameter:enlänge 26

Beispiel:

```
DECL KrlMsgOpt_T krlMsgOpt
INT msgHandle

krlMsgOpt.VL_Stop=TRUE
krlMsgOpt.Clear_P_Reset=TRUE
krlMsgOpt.Log_To_DB=FALSE

msl_FireStringParams(#NOTIFY, „myKxrDataBase“, 12348, "chillDays", krlMsgOpt, msgHandle, "monday",
"friday", "sunday" )

; myKxrDataBase

<uiText key="chillDays">
  <text xml:lang="en-US">I have to chill on %1, %2 and %3 anyway</text>
</uiText>

;Message fired: „I have to chill on monday, friday and sunday anyway“
```

6.5.1.5 msl_FireFrameData()

Beschreibung:

Absetzen einer Meldung mit der Möglichkeit ein Frame als – Parameter zu übergeben.

Funktionskopf:

GLOBAL DEF msl_FireFrameData(msgTyp:IN, msgModule[:IN, msgNo:IN, msgTxt[:IN, krlMsgOpt:IN, msgHandle:OUT, frameData:IN)

Parameter:

Parameter	Datentyp	Beschreibung
msgTyp	Enum	Meldungstyp: • #notify: Hinweismeldung • #state: Zustandsmeldung • #quit: Quittiermeldung • #waiting: Wartemeldung
msgModule[]	CHAR	Absender bzw. Name der kxr- Meldungsdatenbank: • max. Zeichenlänge 26.

msgNo	INT	Meldungsnummer.
msgTxt[]	CHAR	Meldungstext: • max. Zeichenlänge: 80.
krlMsgOpt	Struktur	Meldungsverhalten. Struktur vom Typ KrlMsgOpt_T: • vl_stop ◦ TRUE: Meldungsausgabe löst Vorlaufstopp aus. ◦ FALSE: Kein Vorlaufstopp bei Meldungsausgabe. • clear_p_reset ◦ TRUE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes gelöscht. ◦ FALSE: Meldungen werden bei Abwahl oder zurücksetzen des Programmes nicht gelöscht. • clear_p_SAW ◦ TRUE: Meldungen werden bei Satzanwahl gelöscht. ◦ FALSE: Meldungen werden bei Satzanwahl nicht gelöscht. • log_to_DB ◦ TRUE: Meldungen werden im Ereignisprotokoll geloggt. ◦ FALSE: Meldungen werden nicht geloggt.
msgHandle	INT	Rückgabewert • -1: Die Meldung konnte nicht abgesetzt werden. • 0: Hinweismeldung. • >0: Die Meldung wurde erfolgreich abgesetzt.
frameData	Frame	Frame-Parameter • Die Anzahl initialisierter Komponenten der Frame-Struktur ist beliebig.

Beispiel:

```
INT msgHandle

msl_FireFrameData(#NOTIFY, "Tools", 12348, "Tooldata[1]:", val_st_notMsgOpt, msgHandle,
TOOL_DATA[1])

;Message fired: „Tooldata[1]:{x 10.12, y 20.26, z 54.88, a 45.88, b 98.40, c 82.51 }“
```

6.5.2 Meldungszustand

6.5.2.1 msl_ExistsMsg(...)

Beschreibung:

Prüft ob eine Meldung ansteht.

Funktionskopf:

GLOBAL DEFFCT BOOL msl_ExistsMsg(msgHandle:IN)

Rückgabe:

- TRUE – Meldung steht aktuell an
- FALSE –Meldung steht nicht an

Parameter:

Parameter	Datentyp	Beschreibung
msgHandle	INT	Handle der Meldung, die geprüft werden soll.

Beispiel:

```
INT msgHandle
BOOL bReturn

;fire example message
msl_FireMsg(#STATE, „exampleModule“, 12345, "example state message", val_st_notMsgOpt,
msgHandle, , , , , )

bReturn=msl_ExistsMsg(msgHandle)

//bReturn==TRUE
```

6.5.2.2 msl_ExistsDialog(..)

Beschreibung:

Prüft ob eine Dialogmeldung ansteht.

Funktionskopf:

GLOBAL DEFFCT BOOL msl_ExistsDialog(msgHandle:IN, keySelected:OUT)

Rückgabe:

- TRUE – Dialogmeldung existiert.
- FALSE – Dialogmeldung existiert nicht.

Parameter:

Parameter	Datentyp	Beschreibung
msgHandle	INT	Handle der Dialogmeldung, die geprüft werden soll
keySelected	INT	Nummer der Schaltfläche • 1...7: Schaltfläche mit der Anwender den Dialog bestätigt hat. • 0: Dialog wurde im Programm und nicht durch Anwender beendet.

6.5.2.3 msl_Buffer(..)

Beschreibung:

Lesen des Meldungspuffer und der Anzahl Meldungen im Puffer.

Funktionskopf:

GLOBAL DEF msl_Buffer(msgBuf[:OUT, msgNo:OUT)

Parameter:

Parameter	Datentyp	Beschreibung
msgBuf[]	MsgBuf_T	Meldungspuffer: • type: Meldungstyp (#sys_quit, #usr_State, ...). • nr: Meldungsnummer. • Modul[]. • msg_txt[]. • par_type. • par_txt. • handle. Der Puffer enthält keine Hinweismeldungen.
msgNo	INT	Anzahl Meldungen im Puffer.

Beispiel:

```
BOOL bReturn
INT msgBuffNo
DECL MSGBUF_T msgBuff[100]

bReturn=msl_Clear(-1)

msl_FireMsg(#STATE, „myKxrDataBase“, 12345, "exampleState", msl_st_notMsgOpt,
msl_i_msgHandle, , , , ,)

msl_Buffer(msgBuff[], msgBuffNo)

// msgBuffNo=1

//msgBuff[1]={TYPE #USR_STATE,NR 'B0011000000111001',MODUL[] "myKxrDataBase",MSG_TXT[]
"exampleState",PAR_TYPE1 #EMPTY,PAR_TXT1[] " ",PAR_TYPE2 #EMPTY,PAR_TXT2[] " ",PAR_TYPE3
#EMPTY,PAR_TXT3[] " ",HANDLE 'B00100111100011100'}
```

6.5.3 Meldungen löschen

6.5.3.1 msl_Clear(..)

Beschreibung:

Löschen von Anwendermeldungen. Hinweismeldungen (msgHandle==0) können nicht gelöscht werden.

Funktionskopf:

GLOBAL DEFFCT BOOL msl_Clear(msgHandle:IN)

Rückgabe:

- TRUE – Löschen der Meldung(en) erfolgreich.
- FALSE – Löschen der Meldung(en) fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
msgHandle	INT	Handle der zu löschenden Meldung: • -1: alle Meldungen des Prozesses werden gelöscht • -99: Meldungen aller Prozesse werden gelöscht

Beispiel:

```
INT msgHandle
BOOL bReturn

msl_FireMsg(#STATE, „exampleModule“, 12345, "example state message", val_st_notMsgOpt,
msgHandle, , , , , )

bReturn=msl_Clear(msgHandle)

//bReturn==TRUE
```

6.5.4 Parameterzuweisung

6.5.4.1 *msl_paramInt(..)*

Beschreibung:

Platzhalter einen Integer Wert zuweisen.

Funktionskopf:

GLOBAL DEFFCT KrlMsgPar_T msl_paramInt(paramInt:IN)

Rückgabe:

Struktur vom Typ KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_INT=paramInt.

Parameter:

Parameter	Datentyp	Beschreibung
paramInt	INT	Integer-Wert der einen Platzhalter ersetzt.

6.5.4.2 *msl_paramReal(..)*

Beschreibung:

Platzhalter einen Real Wert zuweisen.

Funktionskopf:

GLOBAL DEFFCT KrlMsgPar_T msl_paramReal(paramReal:IN)

Rückgabe:

Struktur vom Typ KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_REAL=paramReal.

Parameter:

Parameter	Datentyp	Beschreibung
paramReal	REAL	Real-Wert der einen Platzhalter ersetzt

6.5.4.3 *msl_paramBool(..)*

Beschreibung:

Platzhalter einen bool'schen Wert zuweisen.

Funktionskopf:

GLOBAL DEFFCT KrlMsgPar_T msl_paramBool(paramBool:IN)

Rückgabe:

Struktur vom Typ KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_BOOL=paramBool.

Parameter:

Parameter	Datentyp	Beschreibung
paramBool	BOOL	Bool'scher-Wert der einen Platzhalter ersetzt

6.5.4.4 *msl_paramString(..)*

Beschreibung:

Platzhalter einer Zeichenkette zuweisen.

Funktionskopf:

GLOBAL DEFFCT KrlMsgPar_T msl_paramString(paramString[:IN])

Rückgabe:

Struktur vom Typ KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_TXT=paramString[.]

Parameter:

Parameter	Datentyp	Beschreibung
paramString	CHAR	String der einen Platzhalter ersetzt.

6.6 FileLib: Dateien

6.6.1 Daten auf Festplatte schreiben

Schreiben von Daten oder Text in Dateien auf Festplatte. Jeder Eintrag wird fortlaufend nummeriert und erhält einen Zeitstempel.

6.6.1.1 *fil_LogValuesToCsv(...)*

Beschreibung:

Schreiben von Daten im csv – Format auf HDD/SSD. Die Datei wird in das Verzeichnis „C:\KRC\ROBOTER\UserFiles“ geschrieben.

Die Zeichenlänge einer Zeile ist auf 1024 Zeichen beschränkt. Es werden maximal 10 Dateien mit jeweils 1000 Zeilen geschrieben. Im Anschluß werden in einem Ringpuffer die Dateien überschrieben.

Syntax Dateiname: [fileName]_[1...10].csv

Die Csv-Dateien werden in einem KRCDiag gesichert.

Funktionskopf:

GLOBAL DEF fil_LogValuesToCsv(fileName[:IN, headLine[:IN, fileCount:OUT, rPar1:IN, rPar2:IN, rPar3:IN, rPar4:IN, rPar5:IN, rPar6:IN, rPar7:IN, rPar8:IN, rPar9:IN, rPar10:IN)

Parameter:

Parameter	Datentyp	Beschreibung
fileName[]	CHAR	Dateiname
headLine[]	CHAR	Kopfzeile. Maximale Länge sind 462 Zeichen
fileCount	fil_csvFileCount_T	Die Struktur enthält Komponenten für die Dateinummer und Zeilennummer. Die Komponenten werden innerhalb der Routine inkrementiert.
rPar[1...10]	REAL	Parameter die geschrieben werden.

Beispiel:

```
;declaration dat-file:
DECL fil_csvFileCount_T valuesToCsvFileCount={fileNo 0, lineNo 0}
DECL CHAR exampleHeadline[128]
exampleHeadline[]="random1;random2;random3;random4;random5;random6;random7;random8;random9;random10"

FOR idx = 1 TO 10
  randVal[1]=mal_rand_LCG($ROB_TIMER)
  WAIT SEC 0.024
```

```

    randVal[2]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[3]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[4]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[5]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[6]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[7]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[8]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[9]=mal_rand_LCG($ROB_TIMER)
    WAIT SEC 0.024
    randVal[10]=mal_rand_LCG($ROB_TIMER)

    fil_LogValuesToCsv("exampleFile", exampleHeadline[], valuesToCsvFileCount, randVal[1],
    randVal[2], randVal[3], randVal[4], randVal[5], randVal[6], randVal[7], randVal[8], randVal[9],
    randVal[10])
    WAIT SEC 0.2
ENDFOR

;exampleFile_1.csv
No;Time;random1;random2;random3;random4;random5;random6;random7;random8;random9;random10
1;2019-06-09T07:28:09; 0.52; 0.74; 0.36; 0.35; 0.23; 0.08; 0.93; 0.82; 0.80; 0.65
2;2019-06-09T07:28:11; 0.92; 0.96; 0.98; 0.96; 0.01; 0.12; 0.04; 0.89; 0.81; 0.83
3;2019-06-09T07:28:12; 0.94; 0.95; 0.87; 0.79; 0.74; 0.98; 0.97; 0.85; 0.83; 0.69
4;2019-06-09T07:28:14; 0.09; 0.98; 0.06; 0.98; 0.89; 0.81; 0.19; 0.14; 0.09; 0.00
5;2019-06-09T07:28:16; 0.28; 0.23; 0.18; 0.03; 0.95; 0.87; 0.01; 0.93; 0.11; 0.03
6;2019-06-09T07:28:17; 0.90; 0.81; 0.70; 0.78; 0.86; 0.78; 0.70; 0.91; 0.82; 0.81
7;2019-06-09T07:28:19; 0.92; 0.94; 0.82; 0.71; 0.59; 0.61; 0.56; 0.44; 0.29; 0.18
8;2019-06-09T07:28:20; 1.00; 0.91; 0.09; 0.98; 0.83; 0.70; 0.58; 0.47; 0.32; 0.33
9;2019-06-09T07:28:22; 0.58; 0.43; 0.35; 0.46; 0.41; 0.39; 0.44; 0.46; 0.34; 0.19
10;2019-06-09T07:28:24; 0.73; 0.62; 0.61; 0.15; 0.67; 0.12; 0.03; 0.11; 0.13; 0.01

```

6.6.1.2 *fil_LogLineToCsv(...)*

Beschreibung:

Schreiben von Zeile in eine csv-Datei auf HDD/SSD. Die Datei wird in das Verzeichnis „C:\KRC\ROBOTER\UserFiles“ geschrieben.

Die Zeichenlänge einer Zeile ist auf 1024 Zeichen beschränkt. Es werden maximal 10 Dateien mit jeweils 1000 Zeilen geschrieben. Im Anschluß werden in einem Ringpuffer die Dateien überschrieben.

Syntax Dateiname: [fileName]_[1...10].csv

Die Csv-Dateien werden in einem KRCDiag gesichert.

Funktionskopf:

GLOBAL DEF fil_LogLineToCsv(fileName[:IN, headLine[:IN, fileCount:OUT , logLine[:IN)

Parameter:

Parameter	Datentyp	Beschreibung
fileName[]	CHAR	Dateiname
headLine[]	CHAR	Kopfzeile. Maximale Länge sind 462 Zeichen
fileCount	fil_csvFileCoun	Die Struktur enthält Komponenten für die Dateinummer und

	t_T	Zeilennummer. Die Komponenten werden innerhalb der Routine inkrementiert.
logLine[]	CHAR	Zeile die in Datei geschrieben werden. Die Zeilenlänge ist auf 438 Zeichen begrenzt.

Beispiel:

```
;declaration dat-file:
DECL fil_csvFileCount_T logLineCsvFileCount={fileNo 0, lineNo 0}
INT idx

FOR idx=1 TO 3
    fil_LogLineToCsv("CsvLogFileName", "MyHeadline", logLineCsvFileCount , „I need to chill“)
    WAIT SEC 1
ENDFOR

; CsvLogFileName_1.csv
No;Time; MyHeadline
1;2018-10-09T09:54:39; I need to chill
2;2018-10-09T09:54:40; I need to chill
3;2018-10-09T09:54:41; I need to chill
```

6.6.1.3 **fil_LogLineToTxt(...)**

Beschreibung:

Schreiben von Daten im txt – Format auf HDD/SSD.

Funktionell identisch mit fil_LogLineToCsv.

6.6.1.4 **fil_LogMessage(...)**

Beschreibung:

Schreiben von Zeichenketten in eine *.log Datei auf HDD/SSD. Die Datei wird in dem Verzeichnis „C:\KRC\ROBOTER\UserFiles“ angelegt. Einer Zeile können 2 integer und 8 real – Parameter übergeben werden. Für die Parameter sind Platzhalter im Text vorzusehen.

Die Zeichenlänge einer Zeile ist auf 1024 Zeichen beschränkt. Es werden maximal 10Dateien mit jeweils 1000 Zeilen geschrieben. Im Anschluß werden in einem Ringpuffer die Dateien überschrieben.

Syntax Dateiname: LogMessage_[1..10].log

Funktionskopf:

```
GLOBAL DEF fil_LogMessage(logLine[]:IN, iPar1:IN, iPar2:IN, rPar3:IN, rPar4:IN, rPar5:IN, rPar6:IN,
rPar7:IN, rPar8:IN, rPar9:IN, rPar10:IN)
```

Parameter:

Parameter	Datentyp	Beschreibung
logLine[]	CHAR	Zeichenkette zum Schreiben in eine Datei. Werden Parameter

		übergeben, sind Platzhalter vorzusehen (%1, %2,...).
iPar[1..2]	INT	Parameter
rPar[3..10]	REAL	Parameter

Beispiel:

```

fil_LogMessage("The error should never occur..."," ", " ", " ", " ", " ", " ")
fil_LogMessage("Base-data from Base no %1 = (X %2,Y %3, Z %4, A %5, B %6, C %7)",
3, ,Base_Data[3].X,Base_Data[3].Y ,Base_Data[3].Z ,Base_Data[3].A ,Base_Data[3].B ,Base_Data[3].C ,
, )

; logmessage_1.log
3;2019-10-09T09:54:40;The error should never occur...
4;2019-10-09T09:54:40;Base-data from Base no 3 = (X 10.00,Y 20.00, Z 30.00, A 30.00, B 20.00, C
10.00)

```

6.6.2 Dateibearbeitung

6.6.2.1 *fil_FileExists(...)*

Beschreibung:

Prüfung, ob eine Datei existiert.

Die Datei wird in dem Verzeichnis „C:\KRC\ROBOTER\UserFiles“ gesucht.

Funktionskopf:

GLOBAL DEFFCT BOOL fil_FileExists(fileName[:IN])

Rückgabe:

- TRUE – Datei existiert.
- FALSE – Datei existiert nicht.

Parameter:

Parameter	Datentyp	Beschreibung
fileName[]	CHAR	Dateiname

Beispiel:

```

BOOL bResult

bResult=fil_FileExists("filenotexists.log")

//bResult==FALSE

;generate logfile
fil_LogMessage("Log file with dummy string",,,,,,,,,)

bResult=fil_FileExists("logmessage_1.log ")

//bResult==TRUE

```

6.6.2.2 *fil_FileRemove(...)*

Beschreibung:

Löschen einer Datei auf HDD.

Die Datei wird in dem Verzeichnis „C:\KRC\ROBOTER\UserFiles“ gesucht und falls vorhanden gelöscht.

Funktionskopf:

GLOBAL DEFFCT BOOL *fil_FileRemove*(fileName[:IN])

Rückgabe:

- TRUE – Operation erfolgreich
- FALSE – Operation fehlgeschlagen

Parameter:

Parameter	Datentyp	Beschreibung
fileName[]	CHAR	Dateiname

Beispiel:

```
BOOL bResult

;generate logfile
fil_LogMessage("Log file with dummy string",,,,,,,,,)

;remove existing file
bResult=fil_FileRemove("logmessage_1.log ")

//bResult==TRUE
```

6.7 StringLib: Zeichenketten

6.7.1 Variablenwerte konvertieren

6.7.1.1 *stl_boolToString(..)*

Beschreibung:

Überführt einen boolschen Wert in eine Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL *stl_boolToString*(valueBool:IN, valueStr[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
valueBool	BOOL	Wert der in einen string konvertiert wird
valueStr[]	CHAR	Wert von valueBool als string

Beispiel:

```
DECL BOOL bSuccess, bVariable
DECL CHAR boolString[5]

bVariable=TRUE

bSuccess=stl_boolToString(bVariable, boolString[])

// boolString[]== "true"
```

6.7.1.2 **stl_intToString(..)**

Beschreibung:

Überführt einen ganzzahligen Wert in eine Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_intToString(valueInt:IN, valueStr[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
valueInt	INT	Wert einer Ganzzahl, der in eine Zeichenkette konvertiert wird.
valueStr[]	CHAR	Wert der Ganzzahl als String

Beispiel:

```
DECL BOOL bSuccess
DECL CHAR intString[11]
INT iNum

iNum=12345

bSuccess=stl_intToString(iNum, intString[])

// intString[]== "12345"
```

6.7.1.3 **stl_realToString(..)**

Beschreibung:

Überführt einen reellen Wert in eine Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_realToString(valueReal:IN, valueStr[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
valueReal	REAL	Wert der in einen String konvertiert wird
valueStr[]	CHAR	Wert von valueReal als Zeichenkette. Der Wert wird auf zwei Kommastellen gerundet. Bei Werten ≥ 100000.0 bzw. ≤ -100000.0 wird die Exponentialschreibweise verwendet.

Beispiel:

```
DECL BOOL bSuccess
DECL CHAR realString[8]
REAL rValue

rValue=12345.6789

bSuccess=stl_realToString(rValue, realString[])

// realString[]== "12345.68"
```

6.7.1.4 stl_frameToString(..)

Beschreibung:

Überführt eine Frame-Struktur in eine Zeichenkette.

Syntax: „{X ...,Y ...,Z ...,A ...,B ...,C ...}“

Funktionskopf:

GLOBAL DEFFCT BOOL stl_frameToString(valueFrame:IN, valueStr[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
valueFrame	Frame	Wert der in einen String konvertiert wird. Nicht initialisierte Komponenten werden mit „0“ vorbelegt.
valueStr[]	CHAR	Wert von valueFrame als String

Beispiel:

```
DECL CHAR frameLine[128]
BOOL bResult
FRAME frameData

frameData.X=657.543
frameData.Y=45.98
frameData.Z=98543.56
frameData.A=20.78
frameData.B=112.65
frameData.C=156.76

bResult=stl_frameToString(FrameData, frameLine[])

//bResult==TRUE
//frameLine[]="{X 657.543,Y 45.98,Z 98543.6,A 20.78,B 112.65 ,C 156.76}"
```

6.7.1.5 stl_axisToString(..)

Beschreibung:

Überführt eine E6AXIS-Struktur in eine Zeichenkette.

Syntax: „{A1 ..., A2 ..., A3 ..., A4 ..., A5 ..., A6 ..., E1 ..., E2 ..., E3 ...,E4 ...,E5 ...,E6 ...}“

Funktionskopf:

GLOBAL DEFFCT BOOL stl_axisToString(valueAxis:IN, valueStr[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
valueAxis	E6AXIS	Struktur die in einen String konvertiert wird. Nicht initialisierte Komponenten werden mit „0“ vorbelegt.
valueStr[]	CHAR	Wert von valueAxis als String

Beispiel:

```
DECL CHAR axisLine[256]
BOOL bResult
E6AXIS E6axisData

E6axisData.A1=120.32
E6axisData.A2=-96.38
E6axisData.A3=45.92
E6axisData.A4=-34.99
E6axisData.A5=23.92456
E6axisData.A6=218.435
E6axisData.E1=5272.9256
E6axisData.E2=1.92
E6axisData.E3=54.73
E6axisData.E4=0.0
E6axisData.E5=0.0
E6axisData.E6=0.0
```

```

bResult=stl_axisToString(E6axisData, axisLine[])

//bResult==TRUE
//axisLine[]={A1 120.32,A2 -96.38,A3 45.92,A4 -34.99,A5 23.9246,A6 218.435,E1 5272.93,E2 1.92,E3
54.73,E4 0,E5 0,E6 0}"

```

6.7.1.6 **stl_dateToString(..)**

Beschreibung:

Überführt das Datum einer Datumsstruktur in eine Zeichenkette.

Syntax: „YYYY-MM-DD“

Funktionskopf:

GLOBAL DEFFCT BOOL stl_dateToString(dateStruc:IN, dateTimeString[]):OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
dateStruc	DATE	Struktur vom Typ Date
dateTimeString[]	CHAR	Datum als String. Zeichenlänge >=10

```

DECL CHAR dateTimeString[10]
DECL DATE sampleDate
BOOL bResult

```

```

sampleDate={YEAR 1985, MONTH 6, DAY 12, HOUR 17, MIN 33, SEC 45}

```

```

bResult=stl_dateToString(sampleDate, dateTimeString[])

```

```

//bResult==TRUE
//dateTimeString[]="1985-06-12"

```

```

bResult=stl_dateToString($DATE, dateTimeString[])

```

```

//bResult==TRUE
//dateTimeString[]= → current date

```

6.7.1.7 **stl_timeToString(..)**

Beschreibung:

Überführt die Zeit einer Datumsstruktur in eine Zeichenkette.

Syntax: „HH.MM.SS.“

Funktionskopf:

GLOBAL DEFFCT BOOL stl_timeToString(dateStruc:IN, timeString[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
dateStruc	DATE	Struktur vom Typ Date
timeString[]	CHAR	Zeit als String. Zeichenlänge >=10

```
DECL CHAR timeString[8]
DECL DATE sampleDate
BOOL bResult
```

```
sampleDate={YEAR 1985, MONTH 6, DAY 12, HOUR 17, MIN 33, SEC 45}
```

```
bResult=stl_timeToString(sampleDate, timeString[])
```

```
//bResult==TRUE
//timeString[]="17:33:45"
```

```
bResult=stl_timeToString($DATE, timeString[])
```

```
//bResult==TRUE
//timeString[]= → current time
```

6.7.1.8 stl_dateTimeToString(..)

Beschreibung:

Überführt eine Datumsstruktur in eine Zeichenkette nach ISO-8601 Norm.

Syntax: „YYYY-MM-DDTHH.MM.SS.“

Funktionskopf:

GLOBAL DEFFCT BOOL stl_dateTimeToString(dateStruc:IN, dateTimeString[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
dateStruc	DATE	Struktur vom Typ Date der in einen String konvertiert wird
dateTimeString []	CHAR	Wert von dateStruc als String. Zeichenlänge >=19

```

DECL CHAR dateTimeString[19]
DECL DATE sampleDate
BOOL bResult

sampleDate={YEAR 1985, MONTH 6, DAY 12, HOUR 17, MIN 33, SEC 45}

bResult=stl_dateTimeToString(sampleDate, dateTimeString[])

//bResult==TRUE
//dateTimeString[]="1985-06-12T17:33:45"

bResult=stl_dateTimeToString($DATE, dateTimeString[])

//bResult==TRUE
//dateTimeString[]= → current date

```

6.7.1.9 **stl_stringToInt(..)**

Beschreibung:

Überführt eine ASCII-Zeichenkette in eine Ganzzahl.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_stringToInt(valueStr[:OUT, valueInt:OUT)

Rückgabe:

- TRUE – Konvertierung der ASCII-Zeichenkette in eine Ganzzahl erfolgreich.
- FALSE – Konvertierung der ASCII-Zeichenkette in eine Ganzzahl fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
valueStr[]	CHAR	ASCII-Zeichenkette bestehend aus Vorzeichen und Ziffern
valueInt	INT	vorzeichenbehaftete Ganzzahl

Beispiel:

```

DECL CHAR intLine[11]
BOOL bResult
INT intValue

bResult = StrClear(intLine[])
bResult = StrCopy(intLine[], "-123456")

bResult=stl_asciiToInt(intLine[], intValue)

//bResult==TRUE
//intValue=-123456

```

6.7.1.10 **stl_padLeadingZeros(...)**

Beschreibung:

Überführt den Wert einer Ganzzahl in eine Zeichenkette. Die Zeichenkette wird linksbündig mit Nullen aufgefüllt.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_padLeadingZeros(iValue:IN, padString[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
iValue	INT	Wert einer Ganzzahl, der in eine Zeichenkette konvertiert wird.
padString	CHAR	Wert der Ganzzahl als Zeichenkette mit linksbündig aufgefüllten Nullen

Beispiel:

```
DECL CHAR padString[20]
BOOL bResult

bResult=stl_padLeadingZeros(123456, padString[])
//padString[] == "0000000000000000123456"

bResult=stl_padLeadingZeros(-56, padString[])
//padString[] == "000000000000000000-56"
```

6.7.2 Zeichenketten bearbeiten

6.7.2.1 stl_toUpper(..)

Beschreibung:

Konvertierung einer Zeichenkette in Großbuchstaben.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_toUpper(line[:IN, lineUpper[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette die in Großbuchstaben konvertiert wird
lineUpper[]	CHAR	Zeichenkette in Großbuchstaben. Deklaration der Länge von lineUpper[]

		>= der Länge von line[].
--	--	--------------------------

Beispiel:

```

BOOL bReturn
DECL CHAR lineUpper[32]

bReturn=stl_toUpper("we want to chill on sunday",lineUpper[])

// lineUpper=="WE WANT TO CHILL ON SUNDAY"

```

6.7.2.2 **stl_toLower(..)**

Beschreibung:

Konvertierung einer Zeichenkette in Kleinbuchstaben.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_toLower(line[:IN, lineLower[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette die in Kleinbuchstaben konvertiert wird
lineLower[]	CHAR	Zeichenkette in Kleinbuchstaben. Deklaration der Länge von lineLower[] >= der Länge von line[]

Beispiel:

```

BOOL bReturn
DECL CHAR lineLower[32]

bReturn=stl_toLower("WE WANT TO CHILL ON SUNDAY",lineLower[])

// lineLower=="we want to chill on sunday"

```

6.7.2.3 **stl_replaceString (..)**

Beschreibung:

Ersetzen eines Platzhalters in einer Zeichenkette. Als Platzhalter wird ‚%s‘ verwendet. Es ist nur ein Platzhalter zulässig.

Funktionskopf:

DEFFCT BOOL stl_replaceString(lineMerge[:OUT, lineParam[:IN , strReplace[:IN)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
lineMerge[]	CHAR	Zeichenkette mit ersetzttem Platzhalter
lineParam[]	CHAR	Zeichenkette mit Platzhalter Die Länge darf max. 470 Zeichen betragen.
strReplace	CHAR	Zeichenkette die eingefügt wird.

Beispiel:

```
DECL CHAR lineMerge[50], lineStringPar[16]
BOOL bResult

bResult = StrClear(lineStringPar[]) ;erzeuge leere Zeichenkette
bResult = StrCopy(lineStringPar[], "we want to chill on %s")
bResult=stl_replaceString(lineMerge[], lineStringPar[] , "sunday")

//lineMerge[]=="we want to chill on sunday"
```

6.7.2.4 **stl_replaceParams(..)**

Beschreibung:

Ersetzen von Platzhaltern in einer Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_replaceParams(lineMerge[:OUT], lineParam[:OUT], paramList[:OUT])

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
lineMerge[]	CHAR	Zeichenkette mit ersetzten Platzhaltern
lineParam[]	CHAR	Zeichenkette mit Platzhalter
paramList[]	fil_msgLogParam_T	Parameterliste

Beispiel:

```
DECL fil_msgLogParam T typeParams[3], stringParams[2]
CHAR lineMerge[470], lineParam[470]
BOOL bResult

;merging int-, real-, bool-parameter to string

bResult=stl_intToString(123, typeParams[1].varValue[])
bResult=stl_realToString(456.789, typeParams[2].varValue[])
```

```

bResult=stl_boolToString(TRUE, typeParams[3].varValue[])
bResult = StrCopy(lineParam[], "Integer-parameter=%1, Real-parameter=%2, Bool-parameter=%3")

bResult=stl_replaceParams(lineMerge[], lineParam[], typeParams[] )

//lineMerge[]=="Integer-parameter=123, Real-parameter= 456.79, Bool-parameter=true"

;merging string-parameter to string

bResult = StrCopy(stringParams[1].varValue[], "wednesday")
bResult = StrCopy(stringParams[2].varValue[], "saturday")
bResult = StrClear(lineParam[])
bResult = StrCopy(lineParam[], "we want to chill on %1 and maybe on %2")

bResult=stl_replaceParams(lineMerge[], lineParam[], msgLogParam[] )

//lineMerge[]=="we want to chill on wednesday and maybe on saturday"

```

6.7.2.5 **stl_Trim(..)**

Beschreibung:

Entfernung von Leerzeichen am Anfang oder am Ende einer Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_Trim(line[:IN, lineTrim[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette deren Leerzeichen am Anfang und zum Ende entfernt werden.
lineTrim[]	CHAR	Zeichenkette ohne Leerzeichen am Anfang oder Ende.

Beispiel:

```

DECL CHAR line[64], lineTrim[64]
BOOL bResult

bResult = StrClear(line[])
bResult = StrCopy(line[], "    we want to chill    ")
bResult = stl_Trim(line[], lineTrim[])

// bResult==TRUE
// lineTrim[]=="we want to chill"

```

6.7.2.6 **stl_TrimLeft(..)**

Beschreibung:

Entfernung von Leerzeichen am Anfang einer Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_TrimLeft(line[:IN, lineTrim[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette deren Leerzeichen am Anfang entfernt werden
lineTrim[]	CHAR	Zeichenkette ohne Leerzeichen am Anfang.

Beispiel:

```
DECL CHAR line[64], lineTrimLeft[64]
BOOL bResult

bResult = StrClear(line[])
bResult = StrCopy(line[], "    we want to chill    ")
bResult = stl_TrimLeft(line[], lineTrimLeft[])

// bResult==TRUE
// lineTrimLeft[]=="we want to chill    "
```

6.7.2.7 stl_TrimRight(..)

Beschreibung:

Entfernung von Leerzeichen am Ende einer Zeichenkette.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_TrimRight(line[:IN, lineTrim[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette deren Leerzeichen zu Beginn und zum Ende entfernt werden.
lineTrim[]	CHAR	Zeichenkette ohne Leerzeichen zu Beginn oder Ende.

Beispiel:

```
DECL CHAR line[64], lineTrimRight[64]
BOOL bResult
```

```

bResult = StrClear(line[])
bResult = StrCopy(line[], "    we want to chill    ")
bResult = stl_TrimRight(line[], lineTrim[])

// bResult==TRUE
// lineTrim[]=="    we want to chill"

```

6.7.2.8 **stl_subString(..)**

Beschreibung:

Gibt einen Teil einer Zeichenkette zurück. Beginnend ab einer Anfangsposition mit einer bestimmten Länge.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_subString(line[:IN, startAt:IN, length:IN, subString[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette aus der ein Teil extrahiert wird
startAt	INT	Anfangsposition (>0...<=Länge line[])
length	INT	Länge der Teilzeichenkette (>0...<=Länge line[])
subString[]	CHAR	Teilzeichenkette

Beispiel:

```

DECL CHAR line[64], lineSub[64]
INT strLength, wordLength
BOOL bResult

bResult = StrClear(line[])
bResult = StrCopy(line[], "we want to chill on wednesday and maybe on saturday")
bResult = stl_SubString(line[], 21, 9, lineSub[])

// lineSub[]="wednesday"

strLength=strLen(lineParam[])
wordLength=strLen("saturday")
bResult = stl_SubString(lineParam[], strLength-wordLength+1, wordLength, lineSub[])

// lineSub[]=="saturday"

```

6.7.2.9 **stl_splitString(..)**

Beschreibung:

Trennt eine Zeichenkette in maximal 10 Teilzeichenketten anhand eines Trennzeichens.

Funktionskopf:

stl_splitString(line[:IN, startAt:IN, separator:IN, numSplit:OUT, splitStr1[:OUT, splitStr2[:OUT, splitStr3[:OUT, splitStr4[:OUT, splitStr5[:OUT, splitStr6[:OUT, splitStr7[:OUT, splitStr8[:OUT, splitStr9[:OUT, splitStr10[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette zur Teilung. Die maximal zulässige Länge beträgt 470 Zeichen.
startAt	INT	Anfangsposition (>0...<=Länge line[])
separator	CHAR	Trennzeichen
numSplit	INT	Anzahl der ermittelten Teilzeichenketten
SplitString1[1..10][]	CHAR	Teilzeichenketten

Beispiel:

```
DECL CHAR line[64]
DECL CHAR separator, splitString1[12], splitString2[12], splitString3[12], splitString4[12],
splitString5[12], splitString6[12], splitString7[12], splitString8[12], splitString9[12],
splitString10[12]
INT numSplitStrings
BOOL bResult

bResult = StrClear(line[])
bResult = StrCopy(line[], "we;want;to;chill;on;wednesday;and;maybe;on;saturday")

separator=";"
bResult = stl_SplitString(line[], 1, separator , numSplitStrings, splitString1[], splitString2[],
splitString3[], splitString4[], splitString5[], splitString6[], splitString7[], splitString8[],
splitString9[], splitString10[])

// numSplitStrings == 10

// splitString1[]=="we"
// splitString2[]=="want"
// splitString3[]=="to"
// splitString4[]=="chill"
// splitString5[]=="on"
// splitString6[]=="wednesday"
// splitString7[]=="and"
// splitString8[]=="maybe"
// splitString9[]=="on"
// splitString10[]=="saturday"
```

6.7.2.10 stl_findString(..)

Beschreibung:

Suchen eines Texts innerhalb einer Zeichenkette.

Funktionskopf:

stl_findString(line[:IN, find[:IN, startAt:IN, caseSens:IN, numHits:OUT, indexOfList[:OUT)

Rückgabe:

- TRUE – Operation erfolgreich.
- FALSE – Operation fehlgeschlagen.

Parameter:

Parameter	Datentyp	Beschreibung
line[]	CHAR	Zeichenkette die durchsucht wird
find[]	CHAR	Textfragment nach dem gesucht wird
startAt	INT	Position ab der gesucht wird (1...Länge Zeichenkette)
caseSens	ENUM	Groß-/Kleinschreibung • #CASE_SENS: Groß-/Kleinschreibung wird berücksichtigt • #NOT_CASE_SENS: Groß-/Kleinschreibung wird nicht berücksichtigt
numHits	INT	Anzahl der gefundenen Textfragmente
indexOfList[]	INT	Liste mit den Positionen der gefundenen Textfragmente.

Beispiel:

```
BOOL bResult
INT startAt, indexOfList[10], numHits
DECL CHAR line[64]

bResult = StrClear(line[])
bResult = StrCopy(line[], "we;want;to;chill;on;wednesday;and;maybe;on;saturday")

startAt=1
bResult=stl_findString(line[], ";", startAt, #NOT_CASE_SENS, numHits,indexOfList[])

//numHits==9
//indexOfList[1]=3
//indexOfList[2]=8
//indexOfList[3]=11
//indexOfList[4]=17
//indexOfList[5]=20
//indexOfList[6]=30
//indexOfList[7]=34
//indexOfList[8]=40
//indexOfList[9]=43

startAt=1
bResult=stl_findString(line[], "wednesday", startAt, #NOT_CASE_SENS, numHits, indexOfList[])

//numHits==1
//indexOfList[1]=21

bResult=stl_findString(line[], "WEDNESDAY", startAt, #CASE_SENS, numHits, indexOfList[])

//numHits==0
```

6.7.3 Zeichenketten prüfen

6.7.3.1 `std::charIsNumeric` (..)

Beschreibung:

Prüfung, ob ein Zeichen eine Ziffer darstellt.

Funktionskopf:

GLOBAL DEFFCT BOOL `std::charIsNumeric`(testChar:IN)

Rückgabe:

- TRUE – Das Zeichen stellt eine Ziffer dar.
- FALSE – Das Zeichen stellt keine Ziffer dar.

Parameter:

Parameter	Datentyp	Beschreibung
testChar	CHAR	Zeichen zur Prüfung

Beispiel:

```
DECL CHAR myChar
BOOL bReturn

myChar="8"
bReturn = std::charIsNumeric(myChar)
//bReturn==TRUE

myChar="X"
bReturn = std::charIsNumeric(myChar)
//bReturn==FALSE
```

6.7.3.2 `std::charIsAlphaNumeric` (..)

Beschreibung:

Prüfung, ob ein Zeichen einen Buchstaben oder eine Ziffer darstellt.

Funktionskopf:

GLOBAL DEFFCT BOOL `std::charIsAlphaNumeric`(testChar:IN)

Rückgabe:

- TRUE – Das Zeichen stellt eine Ziffer oder einen Buchstaben dar.
- FALSE – Das Zeichen stellt kein Ziffer und keinen Buchstaben dar.

Parameter:

Parameter	Datentyp	Beschreibung
testChar	CHAR	Zeichen zur Prüfung

Beispiel:

```
DECL CHAR myChar
BOOL bReturn

myChar="8"
bReturn = stl_charIsNumeric(myChar)
//bReturn==TRUE

myChar="X"
bReturn = stl_charIsNumeric(myChar)
//bReturn==TRUE

myChar="-"
bReturn = stl_charIsNumeric(myChar)
//bReturn==FALSE
```

6.7.3.3 stl_charIsLowerCase (..)

Beschreibung:

Prüfung, ob ein Zeichen einen Buchstaben in Kleinschreibung darstellt.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_charIsLowerCase(testChar:IN)

Rückgabe:

- TRUE – Das Zeichen ist ein Kleinbuchstabe.
- FALSE – Das Zeichen ist kein Buchstabe oder ein Großbuchstabe.

Parameter:

Parameter	Datentyp	Beschreibung
testChar	CHAR	Zeichen zur Prüfung

Beispiel:

```
DECL CHAR myChar
BOOL bReturn

myChar="g"
bReturn = stl_charIsLowerCase(myChar)
//bReturn==TRUE

myChar="X"
bReturn = stl_charIsLowerCase(myChar)
//bReturn==FALSE

myChar="-"
bReturn = stl_charIsLowerCase(myChar)
//bReturn==FALSE
```

6.7.3.4 stl_charIsUpperCase (..)

Beschreibung:

Prüfung, ob ein Zeichen einen Buchstaben in Großschreibung darstellt.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_charIsUpperCase(testChar:IN)

Rückgabe:

- TRUE – Das Zeichen ist ein Großbuchstabe.
- FALSE – Das Zeichen ist kein Buchstabe oder ein Kleinbuchstabe.

Parameter:

Parameter	Datentyp	Beschreibung
testChar	CHAR	Zeichen zur Prüfung

Beispiel:

```
DECL CHAR myChar
BOOL bReturn

myChar="g"
bReturn = stl_charIsUpperCase(myChar)
//bReturn==FALSE

myChar="X"
bReturn = stl_charIsUpperCase(myChar)
//bReturn==TRUE

myChar="-"
bReturn = stl_charIsUpperCase(myChar)
//bReturn==FALSE
```

6.7.3.5 stl_stringIsNullOrSpace (...)

Beschreibung:

Prüfung ob eine Zeichenkette leer ist (initialisierte Länge=0) oder ausschließlich aus Leerzeichen besteht.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_stringIsNullOrSpace(testString[]):OUT)

Rückgabe:

- TRUE – Die Zeichenkette ist leer oder besteht ausschließlich aus Leerzeichen
- FALSE – Die Zeichenkette ist nicht leer und besteht nicht ausschließlich aus Leerzeichen

Parameter:

Parameter	Datentyp	Beschreibung
testString[]	CHAR	Zeichenkette zur Prüfung

Beispiel:

```
DECL CHAR myString[64]
BOOL bReturn

bReturn = StrClear(myString[])
bReturn = stl_stringIsNullOrSpace(myString[])
//bReturn==TRUE

bReturn = StrCopy(myString[], "                ")
```

```

bReturn = stl_stringIsNullOrSpace(myString[])
//bReturn==TRUE

bResult = StrCopy(myString[], "1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9")
bReturn = stl_stringIsNullOrSpace(myString[])
//bReturn==FALSE

```

6.7.3.6 *stl_ipAdressValid(...)*

Beschreibung:

Prüfung ob eine Zeichenkette eine gültige IP-Adresse (V4) enthält.

Funktionskopf:

GLOBAL DEFFCT BOOL stl_ipAdressValid(ipAdress[:OUT, iOctet[:OUT)

Rückgabe:

- TRUE – Die Zeichenkette stellt eine valide IP-Adresse dar.
- FALSE – Die Zeichenkette stellt keine valide IP-Adresse dar.

Parameter:

Parameter	Datentyp	Beschreibung
ipAdress[]	CHAR	Zeichenkette zur Prüfung
iOctet[]	INT	Feld für 4Oktetten. Stellt die Zeichenkette keine valide IP-Adresse dar, werden die Oktetten mit -1 belegt.

Beispiel:

```

;Sourcecode
DECL CHAR myIpAddress[15]
INT octets[4]
BOOL bReturn

bReturn = StrClear(myIpAddress[])
bReturn = StrCopy(myIpAddress[], "127.212.31.1", octets[])
bReturn = stl_ipAdressValid(myIpAddress[])
//bReturn==TRUE
//octets[1]=127
//octets[2]=212
//octets[3]=31
//octets[4]=1

bReturn = StrClear(myIpAddress[])
bReturn = StrCopy(myIpAddress[], "i am busy", octets[4])
bReturn = stl_ipAdressValid(myIpAddress[])
//bReturn==FALSE
//octets[1]=-1
//octets[2]=-1
//octets[3]=-1
//octets[4]=-1

```

6.8 Varlib: Variablen und Typen

6.8.1 Typenkonvertierung

6.8.1.1 *val_UintToInt(...)*

Beschreibung:

Konvertierung einer vorzeichenlose Ganzzahl zu einer vorzeichenbehafteten Ganzzahl. Mit dem Parameter „bitSize“ können variable Datenlängen der vorzeichenlosen Ganzzahl (Signalvereinbarung) betrachtet werden. Signalvereinbarungen mit Datenlängen kleiner 32 können auf Basis des Zweierkomplements in eine vorzeichenbehaftete Zahl konvertiert werden.

Funktionskopf:

GLOBAL DEFFCT INT mal_UintToInt(uintValue:IN, bitSize:IN)

Rückgabe:

Vorzeichenbehaftete Ganzzahl.

Parameter:

Parameter	Datentyp	Beschreibung
uintValue	INT	vorzeichenlose Ganzzahl
bitSize	INT	Bitlänge der vorzeichenlosen Ganzzahl (2...31)

Beispiel:

```
INT iResult, bitSize, uintValue

bitSize=4
uintValue=13
iResult=val_UintToInt(uintValue, bitSize)

//iResult=-3
```

6.8.1.2 *val_IntToUint(...)*

Beschreibung:

Konvertierung einer vorzeichenbehafteten Ganzzahl zu einer vorzeichenlose Ganzzahl. Mit dem Parameter „bitSize“ können variable Datenlängen der vorzeichenbehafteten Ganzzahl (Signalvereinbarung) betrachtet werden. Die Konvertierung erfolgt auf Basis des Zweierkomplements.

Funktionskopf:

GLOBAL DEFFCT INT mal_IntToUint(intValue:IN, bitSize:IN)

Rückgabe:

Vorzeichenlose Ganzzahl.

Parameter:

Parameter	Datentyp	Beschreibung
-----------	----------	--------------

intValue	INT	vorzeichenbehaftete Ganzzahl (-1073741824 ...1073741823)
bitSize	INT	Bitlänge der vorzeichenbehafteten Ganzzahl (2...31)

Beispiel:

```
INT iResult, bitSize, intValue

bitSize=4
intValue=-3
iResult=val_IntToUint(intValue, bitSize)

//iResult=13
```

6.8.1.3 **val_bytesToInt(...)**

Beschreibung:

Konvertierung eines vier Elemente großen Byte-Arrays in eine Ganzzahl.

Funktionskopf:

DEFFCT INT val_bytesToInt(Bytes[:OUT])

Rückgabe:

Vorzeichenbehaftete Ganzzahl

Parameter:

Parameter	Datentyp	Beschreibung
Bytes[]	CHAR	Byte-Array der Größe 4.

Beispiel:

```
INT iResult
DECL CHAR Bytes[4]

Bytes[1]=10
Bytes[2]=20
Bytes[3]=30
Bytes[4]=40
iResult=val_bytesToInt(Bytes[])

//iResult=673059850
```

6.8.1.4 **val_shiftBitsRight(...)**

Beschreibung:

Bitverschiebung nach rechts.

Funktionskopf:

DEFFCT INT val_shiftBitsRight(intValue:IN, bitsNo:IN)

Rückgabe:

Ergebnis der Bitverschiebung

Parameter:

Parameter	Datentyp	Beschreibung
intValue	INT	Ganzzahl (1...maxInt)
bitsNo	INT	Anzahl Bits um die verschoben wird. (>0)

6.8.2 Bit-Konvertierung

6.8.2.1 **val_bitConvIntToBuffer(...)**

Beschreibung:

Konvertierung einer Ganzzahl zu einem vier Elemente großen Byte-Array.

Funktionskopf:

GLOBAL DEFECT CHAR[4] val_bitConvIntToBuffer(intValue:IN)

Rückgabe:

Byte-Array vom Datentyp CHAR der Feldgröße 4.

Parameter:

Parameter	Datentyp	Beschreibung
intValue	INT	Ganzzahl zur Konvertierung in ein Byte-Array.

6.8.2.2 **val_bitConvRealToBuffer(...)**

Beschreibung:

Konvertierung einer reellen zu einem vier Elemente großen Byte-Array.

Funktionskopf:

GLOBAL DEFECT CHAR[4] val_bitConvRealToBuffer(realValue:IN)

Rückgabe:

Byte-Array vom Datentyp CHAR der Feldgröße 4.

Parameter:

Parameter	Datentyp	Beschreibung
realValue	REAL	Reelle Zahl zur Konvertierung in ein Byte-Array

6.8.2.3 **val_bitConvBoolToBuffer(...)**

Beschreibung:

Konvertierung eines booleschen Parameters zu einem Byte.

Funktionskopf:

GLOBAL DEFFCT CHAR[1] val_bitConvBoolToBuffer(boolValue:IN)

Rückgabe:

Byte-Array vom Datentyp CHAR der Feldgröße 1.

Parameter:

Parameter	Datentyp	Beschreibung
boolValue	BOOL	Datentyp bool zur Konvertierung in ein Byte-Array

6.8.2.4 **val_bitConvBufferToInt(...)**

Beschreibung:

Konvertierung eines vier Elemente großen Byte-Array zu einer Ganzzahl.

Funktionskopf:

GLOBAL DEFFCT INT val_bitConvBufferToInt(intBuff[:IN])

Rückgabe:

Ganzzahl.

Parameter:

Parameter	Datentyp	Beschreibung
intBuff[]	CHAR	Byte-Array zur Konvertierung in eine Ganzzahl.

6.8.2.5 **val_bitConvBufferToReal(...)**

Beschreibung:

Konvertierung eines vier Elemente großen Byte-Array zu einer reellen Zahl.

Funktionskopf:

GLOBAL DEFFCT INT val_bitConvBufferToReal(realBuff[:IN])

Rückgabe:

Reelle Zahl.

Parameter:

Parameter	Datentyp	Beschreibung
realBuff[]	CHAR	Byte-Array zur Konvertierung in eine reelle Zahl.

6.8.2.6 **val_bitConvBufferToBool(...)**

Beschreibung:

Konvertierung eines ein Element großen Byte-Array zu einer booleschen Variable.

Funktionskopf:

GLOBAL DEFFCT BOOL val_bitConvBufferToBool(boolBuff[]):IN)

Rückgabe:

Boolesche Variable.

Parameter:

Parameter	Datentyp	Beschreibung
boolBuff[]	CHAR	Byte-Array zur Konvertierung in eine boolesche Variable

6.8.2.7 **val_bitConvRealToInt(...)**

Beschreibung:

Konvertierung einer reellen Zahl in ein Byte-Array und vom Byte-Array zu einer Ganzzahl.

Funktionskopf:

GLOBAL DEFFCT INT val_bitConvRealToInt(realValue:IN)

Rückgabe:

Ganzzahl, aus einer reellen Zahl konvertiert.

Parameter:

Parameter	Datentyp	Beschreibung
realValue	REAL	Reelle Zahl zur Konvertierung in eine Ganzzahl

6.8.2.8 **val_bitConvIntToReal(...)**

Beschreibung:

Konvertierung einer Ganzzahl in ein Byte-Array und vom Byte-Array zu einer reellen Zahl.

Es lassen sich nicht alle Ganzzahlen in eine reelle Zahl konvertieren. Im Fehlerfall wird die Meldung „realValue= NaN Wert ungültig“ ausgegeben. Die Konvertierung einer Ganzzahl, die zuvor über die

Funktion `val_bitConvRealToInt()` ermittelt wird (Austausch einer REAL Variable über eine Signalvereinbarung), ergibt grundsätzlich einen gültigen Rückgabewert.

Funktionskopf:

GLOBAL DEFFCT REAL `val_bitConvIntToReal(intValue:IN)`

Rückgabe:

Reelle Zahl, aus einer Ganzzahl konvertiert.

Parameter:

Parameter	Datentyp	Beschreibung
<code>intValue</code>	INT	Ganzzahl zur Konvertierung in eine reelle Zahl

6.8.2.9 `val_bitConvBufferCopy(...)`

Beschreibung:

Kopieren eines Byte-Arrays.

Funktionskopf:

GLOBAL DEF `val_bitConvBufferCopy(destBuffer[:OUT, sourceBuffer[:OUT)`

Rückgabe:

Byte-Array mit den Inhalt der Quelle

Parameter:

Parameter	Datentyp	Beschreibung
<code>destBuffer[]</code>	CHAR	Byte-Array als Ziel
<code>sourceBuffer[]</code>	CHAR	Byte-Array als Quelle

6.8.3 Byte-swapping

6.8.3.1 `val_byteSwap(...)`

Beschreibung:

Vertauschung eines Byte-Array der Größe 4 gemäß little- / big-endian Konvention.

IN[]	OUT[]
1	4
2	3

3	2
4	1

Funktionskopf:

GLOBAL DEFFCT CHAR[4] val_byteSwap(inBytes[:OUT])

Rückgabe:

Byte-Array mit getauschten Elementen.

Parameter:

Parameter	Datentyp	Beschreibung
inBytes[]	CHAR	Byte-Array dessen Elemente getauscht werden.

Beispiel:

```
DECL CHAR inBytes[4], outBytes[4]

inBytes[1]="A"
inBytes[2]="B"
inBytes[3]="C"
inBytes[4]="D"

outBytes[]=val_byteSwap(inBytes[])

// outBytes[] = "DCBA"
```

6.8.3.2 val_byteSwapInteger(...)

Beschreibung:

Vertauschung der Bytes einer Ganzzahl gemäß little- / big-endian Konvention.

Funktionskopf:

GLOBAL DEFFCT INT val_byteSwapInteger(intValue:IN)

Rückgabe:

Ganzzahl nach little- / big-endian konvertiert

Parameter:

Parameter	Datentyp	Beschreibung
intValue	INT	Ganzzahl zur Konvertierung

Beispiel:

```
INT intValue, iResult

intValue=7 ; 'B00000000 00000000 00000000 00000111'
iResult = val_byteSwapInteger(intValue)

//iResult=117440512, 'B00000111 00000000 00000000 00000000'
```

6.8.4 Daten Validierung

6.8.4.1 *val_signalOverflow(...)*

Beschreibung:

Prüfung, ob die Wertzuweisung auf eine Signalvereinbarung den Bereich der Signalvereinbarung überschreitet.

Funktionskopf:

GLOBAL DEFFCT BOOL val_signalOverflow(value:IN, bitSize:IN, numSignum:IN)

Rückgabe:

- TRUE: Wertzuweisung überschreitet Bereich der Signalvereinbarung.
- FALSE: Wertzuweisung innerhalb der Bereichsgrenzen der Signalvereinbarung.

Parameter:

Parameter	Datentyp	Beschreibung
intValue	INT	Ganzzahl zur Konvertierung
bitSize	INT	Breite der Signalvereinbarung
numSignum	ENUM	Darstellung der Signalvereinbarung • #signed: Darstellung vorzeichenbehaftet als Zweierkomplement. • #unsigned: Darstellung als vorzeichenlose Ganzzahl.

Beispiel:

```
INT intValue, bitSize
BOOL bResult

bitSize=8
intValue=-128
bResult = val_signalOverflow(intValue, bitSize, #signed)

//bResult=FALSE

bitSize=8
intValue=-129
bResult = val_signalOverflow(intValue, bitSize, #signed)

//bResult=TRUE

bitSize=8
intValue=255
bResult = val_signalOverflow(intValue, bitSize, #unsigned)

//bResult=FALSE
```

6.8.4.2 *val_validationParamRange(...)*

Beschreibung:

Validierung einer Zahl auf Intervallgrenzen. Im Falle einer Bereichsverletzung wird eine Hinweismeldung ausgegeben.

Funktionskopf:

GLOBAL DEFFCT BOOL val_validationParamRange(paramName[:IN, paramValue:IN,
lowerBoundary:IN, upperBoundary:IN, location:IN)

Rückgabe:

- TRUE: Zahl liegt in dem Bereich.
- FALSE: Zahl liegt nicht in dem Bereich.

Parameter:

Parameter	Datentyp	Beschreibung
paramName[]	Char	Parametername (<=24 Zeichen).
paramValue	Real	Parameter.
lowerBoundary	Real	Untere Grenze des Intervalls.
upperBoundary	Real	Obere Grenze des Intervalls.
location	Enum	Definition, ob die Zahl innerhalb oder außerhalb des Intervalls geprüft werden soll: • #inside: Prüfung, ob sich die Zahl innerhalb des Intervalls befindet. • #outside: Prüfung, ob sich die Zahl außerhalb des Intervalls befindet.

Beispiel:

```
INT value, lowerBoundary, upperBoundary
DECL val_Location_T location
BOOL isValid

value=18
lowerBoundary=19
upperBoundary=21
location=#inside

isValid = val_validationParamRange("ParameterName", value, lowerBoundary, upperBoundary, #INSIDE)

// isValid==FALSE
// Message: „Violation range-parameter:"ParameterName"=18 within (19...21)“
```

6.8.4.3 val_validationParamRelat(...)

Beschreibung:

Validierung einer Zahl gegen einen Vergleichswert. Im Falle einer Verletzung wird eine Hinweismeldung ausgegeben.

Funktionskopf:

GLOBAL DEFFCT BOOL val_validationParamRelat(paramName[:IN, paramValue:IN, relOperator:IN,
compValue:IN)

Rückgabe:

- TRUE: Die Bedingung für den Vergleich ist erfüllt.
- FALSE: Die Bedingung für den Vergleich ist nicht erfüllt.

Parameter:

Parameter	Datentyp	Beschreibung
paramName[]	Char	Parametername (<=24 Zeichen)
paramValue	Real	Parameter
relOperator	Enum	Vergleichsoperator: • #EQUAL, Prüfung auf Gleichheit • #UNEQUAL, Prüfung auf Ungleichheit • #GREATER, Prüfung auf größer • #LESS, Prüfung auf kleiner • #GREATEREQUAL, Prüfung auf größer gleich • #LESSEQUAL, Prüfung auf kleiner gleich
compValue	Real	Vergleichswert

Beispiel:

```
REAL rValue, compValue
DECL val_RelationalOperator_T relOperator
BOOL isValid

rValue=123.654
compValue=654.321
relOperator=#GREATER

isValid = val_validationParamRelat("ParameterName", rValue, relOperator, compValue)

// isValid==FALSE
// Message: „Violation comparison-parameter:“ParameterName“= 123.654 GREATER 654.321“
```

7 Interpreter-Ereignisse

Bereitstellung von Ereignisroutinen des Submit- und Roboter-Interpreter.

7.1 KRL-Module

Die Ereignisroutinen werden in den Modulen:

- \R1\TP\g2Framework\g2Interpreter\g2_subAppl.src (Ereignisse des Submit-Interpreter)
- \R1\TP\g2Framework\g2Interpreter\g2_robAppl.src (Ereignisse des Roboter-Interpreter)

zur Verfügung gestellt.

7.2 Ereignisübersicht

Ereignis	Interpreter	Beschreibung
sub_OnInitCold()	Submit	Aufruf nach Anwahl des Submit-Interpreter.
sub_OnInitHot()	Submit	Aufruf nach Start des Submit-Interpreter der zuvor manuell gestoppt wurde.
sub_OnLoop()	Submit	Zyklischer Aufruf
sub_OnTimerEvent()	Submit	Zyklischer Aufruf nach Ablauf einer Zeitdauer in Sekunden. Die Zeitdauer wird über die Variable sub_i_eventTime konfiguriert.
sub_OnBlockSel()	Submit	Satzanwahl in einem Roboterprogramm.
sub_OnRobProgCancel()	Submit	Aufruf nach Abwahl eines Roboterprogrammes.
sub_OnRobProgReset()	Submit	Aufruf nach Rücksetzen eines Roboterprogrammes.
sub_OnEdgeRising()	Submit	Aufruf bei einer steigenden Flanke der Variable sub_b_edgeEvent.
sub_OnEdgeFalling()	Submit	Aufruf bei einer fallenden Flanke der Variable sub_b_edgeEvent.
sub_OnConfigChanged()	Submit	Aufruf bei Änderungen einer Konfiguration im ApplicationView.
rob_OnProgInit()	Roboter	Programminitialisierung. Die Bearbeitung erfolgt über den Aufruf von rob_ProgInit()
rob_OnProgStart()	Roboter	Aufruf bei Start eines Roboter-Programmes (\$PRO_STATE1 == #P_ACTIVE), sofern sich der Roboter auf der programmierten Bahn befindet. Voraussetzung: rob_ProgInit() wurde zuvor durchlaufen.
rob_OnCycFlagEdgeRising()	Roboter	Aufruf bei positivem Flankenwechsel des \$CYCFLAG[161].

		Voraussetzung: rob_ProgInit() wurde zuvor durchlaufen.
rob_OnCycFlagEdgeFalling()	Roboter	Aufruf bei negativem Flankenwechsel des \$CYCFLAG[161]. Voraussetzung: rob_ProgInit() wurde zuvor durchlaufen.

7.3 Submit-Diagnose

Das Framework stellt in der ApplicationView folgende Diagnosevariablen für den Submit-Interpreter zur Verfügung.

Menü: Anzeige → g2Framework → ApplicationView

Auswahl: Applikation → g2Framework; Komponente → g2_submit

Variable	Beschreibung
sub_c_initColdDate[]	Zeitpunkt des letzten (Kalt-) Start des Submit-Interpreter.
sub_i_CycleCount	Zähler der bei jedem Zyklus des Submit inkrementiert wird.
sub_i_CycleTime	Aktuelle Zykluszeit des Submit.
sub_st_cycleTime.min	Minimale Zykluszeit des Submit
sub_st_cycleTime.max	Maximale Zykluszeit des Submit.

8 Visualisierung

8.1 ApplicationView

The screenshot shows a software interface titled "Application overview". It features two dropdown menus at the top: "Application" (set to "ExampleApplication") and "Component" (set to "GeneralConfig"). Below these is a table with four columns: "Name", "Value", "Unit", and "Upd". The table lists various variables, including integers, booleans, real numbers, strings, and time measurements. The row for "exp_i_intVar1" is highlighted in blue. To the right of the table are four buttons: "Update", "Value", "Save", and "Description". At the bottom, there are three tabs: "Variables" (selected), "Inputs", and "Outputs".

Name	Value	Unit	Upd
exp_i_compNo	3	-	
exp_b_boolVar1	FALSE	-	
exp_i_intVar1	7	-	
exp_r_realVar1	1.234	-	
exp_c_String1[]	"time to chill"	-	
exp_en_dayOfWeekend1	#sun	-	
exp_st_timeMeas1	{exp_timeMe t_start 123, t_stop 321 }	ms	
exp_st_timeMeas1.t_start	123	ms	
exp_st_timeMeas1.t_stop	321	ms	
exp_i_varNotDef	n.d.	-	
exp_i_varNotInit	n.i.	-	

8.1.1 Funktionalität

Das Plugin „ApplicationView“ bietet die Funktionalitäten:

- Anzeige und Änderungen der Zustände von KRL-Variablen.
- Anzeige von Ein- und Ausgängen
- Ändern von Ausgängen bzw. Signalvereinbarungen auf Ausgänge.
- Zyklische Aktualisierung

- Beschreibungsmöglichkeit der Eingänge, Ausgänge und Variablen
- Zuordnung von Variablen, Ein- und Ausgängen zu Applikationen und deren Komponenten über Konfigurationsdateien.
- Definition von Grenzwerten bei Variablen.

Eine Beispielkonfiguration ist Bestandteil der Installation.

8.1.2 Konfiguration

Die Konfiguration einer Applikation besteht aus zwei Dateien:

- Applikationsdefinition
- Beschreibungsdatei (optional)

8.1.2.1 Applikationsdefinition

Syntax Dateinamen: AppViewConfig_{Applikationsname}.xml

Zielverzeichnis auf der Steuerung: C:\KRC\TP\g2Framework\AppViewConfig\

Hinweis: Die Datei wird bei Aufruf von ApplicationView gelesen

Applikationsname:

```
<ApplicationInfo Name="SpotWelding" ...
```

Komponentenname:

```
<DeviceInfo Name="WeldController"
```

Auswahlbedingung für die Komponente:

```
<DeviceInfo ... Trigger="sw_i_maxGun" Operator=">" Value="3">
```

Über die Attribute Trigger, Operator, Value kann gesteuert werden, ob die Komponenten dem Anwender zur Auswahl zur Verfügung gestellt wird.

Trigger:

Global deklarierte KRL-Variable vom Datentyp integer.

Operator:

Es sind drei Operatoren zulaessig

- „=" falls Trigger dem Value entspricht, wird die Komponente zur Auswahl angeboten.
- „<“ falls Trigger kleiner Value ist, wird die Komponente zur Auswahl angeboten.
- „>“ falls Trigger groesser Value ist, wird die Komponente zur Auswahl angeboten.

Value:

Vergleichswert für die Triggervariable.

Variablendefinition

```
<VariableInfo Name="sw_i_maxGun" Min="1" Max="6" Unit=""  
Selection="" Config="User" />
```

Variablenname

```
<VariableInfo Name="sw_i_maxGun" ... />
```

Bereichsgrenzen

```
<VariableInfo ... Min="1" Max="6" ... />
```

Einheit

```
<VariableInfo ... Unit="mm" ... />
```

Auswahl (Datentyp Enum)

```
<VariableInfo ... Selection="#MON,#TUE,#WED,#THU,#FRI" />
```

Benutzergruppe

```
<VariableInfo ... Config="User" />
```

- Config="User": Benutzergruppe Anwender kann Variablenzustände ändern
- Config="Expert": Benutzergruppe Experte kann Variablenzustände ändern
- Config="NoConfig": Änderungen von Variablenzuständen sind nicht möglich

Definition Ein- und Ausgänge

Eingangs-/Ausgangsnamen:

```
<InputInfo Name="sw_di_controllerReady" ... />  
<OutputInfo Name="sw_do_startWeld" ... />
```

Bei den Namen von digitalen Ein- und Ausgängen handelt es sich um integer-Variablen, die als Index für den Ein- bzw. Ausgang verwendet werden.

Signalvereinbarungen müssen global deklariert sein.

Einheit:

```
<InputInfo ... Unit="mm" ... />
```

Bei Signalvereinbarungen wird die Einheit in Anzeige nicht dargestellt.

Bereichsgrenzen:

Bereichsgrenzen werden bei Signalvereinbarungen auf Ausgänge geprüft. Der Maximalwert ist durch die Signalbreite begrenzt. Ist kein max-Wert konfiguriert, wird der durch die Signalbreite ergebene Wert herangezogen.

```
<OutputInfo ... Min="1" Max="6" ... />
```

Benutzergruppe:

```
<OutputInfo ... Config="User" />
```

- Config="User": Benutzergruppe Anwender kann EA-Zustände ändern
- Config="Expert": Benutzergruppe Experte kann EA-Zustände ändern
- Config="NoConfig": Änderungen von EA-Zuständen sind nicht möglich

8.1.2.2 *Beschreibungsdatei*

Hier werden Beschreibungen der Variablen definiert. Die Beschreibungen können mehrsprachig hinterlegt werden. Die Datei ist optional.

Syntax Dateinamen: AppViewConfig_{Applikationsname}.kxr

Verzeichnis auf der Steuerung: C:\KRC\TP\g2Framework\Kxr\

Die Datei wird nach einem Kaltstart der Steuerung gelesen

Applikationsname:

Der Name muss identisch zum Dateinamen (ohne die Endung „.kxr“) sein.

```
<module name="AppViewConfig_SpotWelding">
```

Variablen-, Signalnamen:

```
<uiText key="sw_go_WeldForce">
```

```
...
```

```
</uiText>
```

Beschreibung:

```

<uiText key="sw_go_GunForce">
  <text xml:lang="de-DE">Kraftvorgabe an die Zangensteuerung</text>
  <text xml:lang="en-US">Commanded Force to the gun controller</text>
  <text xml:lang="es-ES">Fuerza nominal al control de armas</text>
</uiText>

```

Die Beschreibung kann in allen Sprachen, die von der HMI unterstützt werden, erweitert werden.

8.1.2.3 Variablendeklaration in KRL

KRL-Variablen und Signalvereinbarungen müssen global deklariert sein.

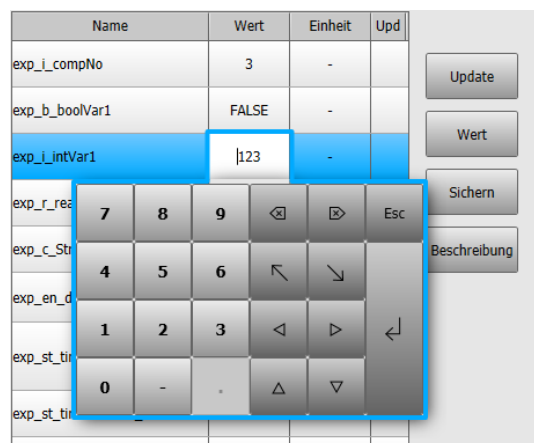
8.1.3 Bedienung

8.1.3.1 Aufruf

Hauptmenü: Anzeige → g2Framework → ApplicationView

8.1.3.2 Variablenwert ändern

- Reiter „Variablen“
- Variable selektieren
- Button „Wert“
- Wert der Variable selektieren
- Wert setzen
- Button „Sichern“



8.1.3.3 Variablenwert zyklisch aktualisieren

- Reiter „Variablen“
- Variable selektieren
- Button „Update“

Name	Wert	Einheit	Upd	
exp_i_compNo	3	-		Update Wert
exp_b_boolVar1	FALSE	-		
exp_i_intVar1	123	-		

Das Zeichen  symbolisiert die zyklische Aktualisierung.

Es können maximal 10 Variablen zeitgleich aktualisieren. Ein- und Ausgänge werden grundsätzlich zyklisch aktualisiert.

8.1.3.4 Variablenbeschreibung

- Reiter „Variablen“
- Variable selektieren
- Button „Beschreibung“

