

www.g2robotics.de

g2Framework
KUKA System Software

Jul 11, 2025

Inhaltsverzeichnis

1	Introduction.....	1
1.1	No warranty.....	1
1.2	Report an error.....	1
2	License.....	2
3	Requirements.....	3
4	Installation.....	4
4.1	KRL-Libraries.....	4
4.2	g2Framework.kop.....	4
4.2.1	Installation on the robot controller.....	4
4.2.2	Installation with WorkVisual (recommended).....	5
5	System variables.....	6
5.1	Cycflags.....	6
5.2	Interrupts - Submitinterpreter.....	6
5.3	Interrupts - Robotinterpreter.....	6
6	Libraries.....	7
6.1	MathLib: Mathematical Functions.....	7
6.1.1	Check for even and odd numbers.....	7
6.1.1.1	mal_isEven(...).	7
6.1.1.2	mal_isOdd(...).	8
6.1.2	Maximum and minimum functions.....	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	Test for equality and interval.....	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	Rounding functions.....	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	Elementary functions.....	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-operations.....	19
6.1.6.1	mal_bitState(...).	19
6.1.6.2	mal_shiftBitsRight(...).	20
6.1.7	Sorting algorithms.....	20
6.1.7.1	mal_quickSort(...).	20
6.1.8	Vectors.....	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 Matrices.....	26
6.1.9.1 mal_rotMatToRPY (...)	27
6.1.10 Geometry.....	27
6.1.10.1 mal_circumference(...)	28
6.1.10.2 mal_circularArc(...)	28
6.1.11 Random numbers.....	29
6.1.11.1 mal_randLcg(...)	29
6.1.11.2 mal_randLcgRange(...)	29
6.2 PhysLib: Physical functions.....	30
6.2.1 Temperature.....	30
6.2.1.1 phl_linThermalExpansion(...)	30
6.2.1.2 phl_CelsiusToKelvin(...)	30
6.2.1.3 phl_KelvinToCelsius(...)	31
6.2.1.4 phl_CelsiusToFahrenheit(...)	31
6.2.1.5 phl_FahrenheitToCelsius(...)	32
6.2.1.6 phl_KelvinToFahrenheit(...)	33
6.2.1.7 phl_FahrenheitToKelvin(...)	33
6.2.2 Time.....	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: Robot specific features.....	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 Test for equality.....	43
6.3.2.1 rol_FrameCompare(...)	43
6.3.3 Distances.....	44
6.3.3.1 rol_FrameDistance(...)	44
6.3.3.2 rol_cartPointDistance(...)	44
6.3.4 Orientations.....	45
6.3.4.1 rol_toolOriToGravity(...)	45
6.3.4.2 rol_posDirToGravity(...)	45

6.3.5 Timer functions.....	46
6.3.5.1 rol_elapsedMsecRobTimer(...)	46
6.3.6 Roboter-Stop.....	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: Axis-specific functions.....	49
6.4.1 Positions.....	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 Distances.....	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 Velocities.....	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 Torques.....	61
6.4.4.1 axl_axisTorqueInfo(...)	61
6.4.5 Transmission.....	61
6.4.5.1 axl_linRatioSpindleDrive(...)	61
6.4.6 Axis state.....	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 Movements.....	66
6.4.7.1 axl_exAxAsyncMovement(...)	66
6.5 MsgLib: User messages.....	67
6.5.1 Fire messages.....	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 Message status.....	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 Clear messages.....	76
6.5.3.1 msl_Clear(..).	76
6.5.4 Parameter assignment.....	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: Handling of Files.....	79
6.6.1 Write data to disk.....	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 File handling.....	82
6.6.2.1 fil_FileExists(...).	82
6.6.2.2 fil_FileRemove(...).	82
6.7 StringLib: Handling of character strings.....	83
6.7.1 Convert variable values.....	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 Edit strings.....	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(..).	94
6.7.2.9 stl_splitString(..).	95
6.7.2.10 stl_findString(..).	96
6.7.3 Check strings.....	97
6.7.3.1 stl_charIsNumeric (..).	97
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(...)	100
6.8	Varlib: Variables and Types	101
6.8.1	Type conversion	101
6.8.1.1	val_UintToInt(...)	101
6.8.1.2	val_IntToUint(...)	102
6.8.1.3	val_bytesToInt(...)	102
6.8.1.4	val_shiftBitsRight(...)	103
6.8.2	Bit-conversion	103
6.8.2.1	val_bitConvIntToBuffer(...)	103
6.8.2.2	val_bitConvRealToBuffer(...)	104
6.8.2.3	val_bitConvBoolToBuffer(...)	104
6.8.2.4	val_bitConvBufferToInt(...)	105
6.8.2.5	val_bitConvBufferToReal(...)	105
6.8.2.6	val_bitConvBufferToBool(...)	105
6.8.2.7	val_bitConvRealToInt(...)	106
6.8.2.8	val_bitConvIntToReal(...)	106
6.8.2.9	val_bitConvBufferCopy(...)	106
6.8.3	Byte-swapping	107
6.8.3.1	val_byteSwap(...)	107
6.8.3.2	val_byteSwapInteger(...)	108
6.8.4	Data validation	108
6.8.4.1	val_signalOverflow(...)	108
6.8.4.2	val_validationParamRange(...)	109
6.8.4.3	val_validationParamRelat(...)	110
7	Interpreter-Events	112
7.1	KRL-Modules	112
7.2	Event Summary	112
7.3	Submit-Diagnosis	113
8	Visualization	114
8.1	ApplicationView	114
8.1.1	Functionality	114
8.1.2	Configuration	115
8.1.2.1	Application definition	115
8.1.2.2	Description file	117
8.1.2.3	Variable declaration in KRL	118
8.1.3	Operation	118
8.1.3.1	Menu	118
8.1.3.2	Change variable value	118
8.1.3.3	Update tag value cyclically	118
8.1.3.4	Variable description	119

1 Introduction

The g2Framework is a collection of KRL routines and a plugin for configuring applications.

The routines cover a large range of robot control functions.

The use of the routines and plugins are described in this documentation.

1.1 No warranty

The software described in this document is not guaranteed. It is the user's responsibility to test the routines for functionality and security. See the BSD license for more information.

1.2 Report an error

If errors occur when using the routines, please report them **info@g2robotics.de**.

The error report should contain the following information:

- Version of the KRL module
(Main Menu- → Display → g2Framework → ApplicationView:
Application: g2Framework, component: g2GlobalLibs)
- Version of the KUKA basic software
- A description of the error
- A program example for error simulation

2 License

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 Requirements

The routines and functionalities are created and tested under KSS8.6.

The following platforms are supported:

- 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-Libraries

The KRL libraries "g2GlobalLibs" can be copied to the \R1 directory on the robot controller. Depending on the platform, the copy process requires the "Expert" user group.

Alternatively, the libraries can be copied to the controller via "WorkVisual":

Method:

- Network connection between WorkVisual and the robot controller.
- WorkVisual → Working area: programming and diagnostics.
- Cell view: Select control.
- KRC Explorer: Right click on the controller.
- Context menu: „Establish control stand“.
- Drag & drop the "g2GlobalLibs" module into the \R1 directory.
- KRC Explorer: Right click.
- Context menu: "Transfer changes".

4.2 g2Framework.kop

4.2.1 Installation on the robot controller

Requirements:

- g2Framework.kop on a USB stick on the robot controller.
- "Expert" user group.

Method:

- Main menu: Commissioning → Additional software.
- Button: „New Software“.
- Button: „Configuration“.
- Button „Path selection“.
- Select USB-Stick and Button „Save“.
- Button „Save“.
- „InstallTech-Auswahl“ select g2Framework.
- Button „Install“.

4.2.2 Installation with WorkVisual (recommended)

Requirement:

- Network connection between WorkVisual and the robot controller.
- "Expert" user group.

Method:

- Installation g2Framework.kop in WorkVisual (KUKA documentation).
- Load the current project from the robot controller.
- Add g2Framework to the project under "Options"..
- Transfer the project to the robot controller and activate it.

5 System variables

Overview of the system variables used in the g2 framework.

5.1 Cycflags

Number	Name	Declaration	Description
160	val_cy_msgLibWait	g2_varLib.dat	Condition for waiting message.
161	val_cy_edgeMonitor	g2_varLib.dat	Monitoring in the robot interpreter

5.2 Interrupts - Submitinterpreter

Prio	Name	Declaration	Description
101	val_int_subLineSel	g2_varLib.dat	Interrupt at block selection
102	val_int_subInitHot	g2_varLib.dat	Interrupt after warm start of the submit interpreter

5.3 Interrupts - Robotinterpreter

Prio	Name	Declaration	Description
100	val_int_robProgStart	g2_varLib.dat	Interrupt at program start (\$PRO_STATE1==#P_ACTIVE)
101	val_int_robCycRising	g2_varLib.dat	Interrupt on rising edge \$CYCFLAG[val_cy_edgeMonitor]
102	val_int_robCycFalling	g2_varLib.dat	Interrupt on falling edge \$CYCFLAG[val_cy_edgeMonitor]

6 Libraries

Overview:

Library	Description
g2_mathLib	Mathematical functions
g2_physLib	Physical functions
g2_robLib	Robot specific features
g2_axisLib	Axis-specific functions
g2_msgLib	Message outputs
g2_fileLib	Writing data or text to files
g2_stringLib	Editing of character strings
g2_varLib	Handling of variables and types

6.1 MathLib: Mathematical Functions

6.1.1 Check for even and odd numbers

6.1.1.1 *mal_isEven(...)*

Description:

Check if an integer is even.

Function header:

GLOBAL DEFFCT BOOL mal_isEven(n:IN)

Return value:

- True: Value is even
- False: Value is odd

Parameter:

Parameter	Data type	Description
n	INT	Value that is checked to see if it is even.

Example:

```
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(...)**

Description:

Check if an integer is odd.

Function header:

GLOBAL DEFFCT BOOL mal_isOdd(n:IN)

Return value:

- True: Value is odd
- False: Value is even

Parameter:

Parameter	Data type	Description
n	INT	Value that is checked to see if it is odd.

Example:

```
INT intValue
BOOL bReturn

intValue=8
bReturn = mal_isOdd(intValue)

//bReturn==FALSE

intValue=45367
bReturn = mal_isOdd(intValue)

//bReturn==TRUE
```

6.1.2 Maximum and minimum functions

6.1.2.1 **mal_maxInt(...)**

Description:

Determination of the maximum of two integers.

Function header:

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

Return value:

Maximum of two integer numbers.

Parameter:

Parameter	Data type	Description
n1	INT	Comparison value 1
n2	INT	Comparison value 2

Example:

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

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

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

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

6.1.2.2 mal_minInt(...)

Description:

Determination of the minimum of two integers.

Function header:

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

Return value:

Minimum of two integer numbers.

Parameter:

Parameter	Data type	Description
n1	INT	Comparison value 1
n2	INT	Comparison value 2

Example:

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

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

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

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

6.1.2.3 mal_maxReal(...)

Description:

Determination of the maximum of two real numbers.

Function header:

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

Return value:

Maximum of two real numbers.

Parameter:

Parameter	Data type	Description
x1	REAL	Comparison value 1
x2	REAL	Comparison value 2

Example:

```
REAL rValue1, rValue2, rMax

rValue1=41.654
rValue2=67.256

rMax=mal_maxReal(rValue1, rValue2)

//rMax==67.256
```

6.1.2.4 mal_minReal(...)

Description:

Determination of the minimum of two real numbers.

Function header:

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

Return value:

Minimum of two real numbers.

Parameter:

Parameter	Data type	Description
x1	REAL	Comparison value 1
x2	REAL	Comparison value 2

Example:

```
REAL rValue1, rValue2, rMin

rValue1=41.654
rValue2=-67.256

rMin=mal_minReal(rValue1, rValue2)

//rMin==-67.256
```


6.1.3 Test for equality and interval

6.1.3.1 *mal_realComp(...)*

Description:

This function determines whether x and y are approximately equal to a relative precision. If x and y are within this interval, they are considered approximately equal and the function returns TRUE. Otherwise, the function returns FALSE. If no value is passed for the interval limit, the function calculates with a limit of 6E-8.

Function header:

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

Return value:

- True: The difference of the real numbers x1, x2 lies within the interval (0...comp)
- False: The difference of the real numbers x1, x2 lies outside the interval (0...comp)

Parameter:

Parameter	Data type	Description
x1	REAL	Comparison value 1
x2	REAL	Comparison value 2
comp	REAL	Interval limit

Example:

```
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 (...)*

Description:

Validation of a number on interval boundaries.

When checking within limits, the completed interval applies:

→ (lowerBoundary <= value <=upperBoundary).

In the case of an out-of-bounds check, the open interval applies:

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

Function header:

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

Return value:

- True: There is no range violation.
- False: There is a range violation.

Parameter:

Parameter	Data type	Description
value	Real	Number for testing
lowerBoundary	Real	Lower limit of the interval. (lowerBoundary<=upperBoundary)
upperBoundary	Real	Upper limit of the interval (upperBoundary>=lowerBoundary)
location	Enum	Definition of whether the number should be checked inside or outside the interval. • #inside: Checking whether the number is within the interval. • #outside: Checking whether the number is outside the interval.

Example:

```
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(...)

Description:

Validating a number against a comparison value.

Function header:

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

Return value:

- True: condition is met.
- False: condition is not met.

Parameter:

Parameter	Data type	Description
value	Real	Number for testing
relOperator	Enum	Comparison operator • #EQUAL, check for equality • #UNEQUAL, check for inequality • #GREATER, check for larger • #LESS, check for smaller • #GREATEREQUAL, check for greater than or equal to • #LESSEQUAL, check for less than or equal to
compValue	Real	Comparison value

Example:

```
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 Rounding functions

6.1.4.1 *mal_floor(...)*

Description:

The function finds the largest integer that is less than or equal to x.

Function header:

GLOBAL DEFFCT INT mal_floor(x :IN)

Return value:

X rounded down to the next smaller integer.

Parameter:

Parameter	Data type	Description
x	REAL	Value rounded down to an integer. (-2147483648.0...2147483647.0)

Example:

```
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(...)*

Description:

The function finds the smallest integer that is greater than or equal to x.

Function header:

GLOBAL DEFFCT INT mal_ceil(x :IN)

Return value:

X rounded up to the next larger integer.

Parameter:

Parameter	Data type	Description
x	REAL	Value rounded up to an integer. (-2147483648.0...2147483647.0)

Example:

```

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(...)**

Description:

Rounds to the nearest integer towards 0.

Function header:

GLOBAL DEFFCT INT mal_trunc(x :IN)

Return value:

Integer with no decimal place of x.

Parameter:

Parameter	Data type	Description
x	REAL	Value rounded to the nearest integer towards 0. (-2147483648.0...2147483647.0)

Example:

```
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 Elementary functions

6.1.5.1 **mal_modulo (...)**

Description:

Calculates the remainder from a division of two integers.

Function header:

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

Return value:

Remainder of a division of two integers.

Parameter:

Parameter	Data type	Description
numerator	INT	Dividend.
denominator	INT	Divisor.

Example:

```
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 (...)

Description:

Power function with integer base 2 exponents

Function header:

GLOBAL DEFFCT INT mal_2pow(exponent:IN)

Return value:

Power value.

Parameter:

Parameter	Data type	Description
exponent	UINT	Exponent

Example:

```
INT iResult

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

6.1.5.3 **mal_powInt (...)**

Description:

Power function with integer exponents and real base.

Function header:

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

Return value:

Power value

Parameter:

Parameter	Data type	Description
base	REAL	Base
exponent	INT	Exponent

Example:

```
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 (...)**

Description:

Power function with unsigned, integer exponents and real base.

Function header:

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

Return value:

Power value

Parameter:

Parameter	Data type	Description
base	REAL	Base
uExponent	INT	Unsigned exponent

Example:

```
REAL result, base
INT exponent

base=7.8
exponent=4

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

6.1.5.5 *mal_hypot (...)*

Description:

Square root of the sum of two squares. Pythagorean theorem.

Function header:

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

Return value:

Hypotenuse of a right triangle with sides x and y.

Parameter:

Parameter	Data type	Description
x	REAL	Side length of the 1st square
y	REAL	Side length of the 2st square

Example:

```
REAL x, y, result

x=3.0
y=4.0

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

6.1.5.6 *mal_hypot3(...)*

Description:

Square root of the sum of three squares.

Function header:

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

Return value:

Side length of a square that results from the sum of three squares.

Parameter:

Parameter	Data type	Description
x	REAL	Side length of the 1st square.
y	REAL	Side length of the 2st square.
z	REAL	Side length of the 3st square.

Example:

```
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-operations

6.1.6.1 *mal_bitState(...)*

Description:

Checking the status of a bit of an integer.

Function header:

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

Return value:

- True: Bit is set.
- False: Bit is not set.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer to check.
bitNo	INT	Number of the bit whose status is checked (0...31).

Example:

```
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(...)*

Description:

Logical right bit shift of an integer. Low-order bits are discarded.

Function header:

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

Return value:

Bit shift result.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer whose bits are being shifted. (≥ 0)
bitsNo	INT	Number of bits to shift.

Example:

```
INT intValue, bitsNo, iResult

intValue=23
bitsNo=2

iResult=mal_shiftBitsRight(intValue, bitsNo)

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

6.1.7 Sorting algorithms

6.1.7.1 *mal_quickSort(...)*

Description:

Quicksort Algorithm. Sorting the elements of a vector in ascending order.

Function header:

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

Return value:

Vector, with the elements sorted in ascending order.

Parameter:

Parameter	Data type	Description
-----------	-----------	-------------

vector[]	REAL	One-dimensional vector.
n	INT	Number of elements of the vector.

Example:

```
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 Vectors

6.1.8.1 *mal_vectorLength(...)*

Description:

Absolute value of a vector.

Function header:

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

Return value:

Betrag des Vektors

Parameter:

Parameter	Data type	Description
vector[]	REAL	One-dimensional vector.
n	INT	Number of elements of the vector.

Example:

```
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(...)*

Description:

Vector product of two vectors in 3-dimensional space.

Function header:

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

Return value:

Vektorprodukt.

Parameter:

Parameter	Data type	Description
vector_a[]	REAL	Three-dimensional vector a
vector_b[]	REAL	Three-dimensional vector b
vectorProduct[]	REAL	Vector product: vector_a[] x vector_b[]

Example:

```
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(...)*

Description:

Spatial product of three vectors in 3-dimensional space. $\vec{a} * (\vec{b} \times \vec{c})$

Function header:

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

Return value:

Spatial product.

Parameter:

Parameter	Data type	Description
vector_a[]	REAL	Three-dimensional vector a
vector_b[]	REAL	Three-dimensional vector b
vector_c[]	REAL	Three-dimensional vector c

Example:

```
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(...)

Description:

Angle between two vectors in 3-dimensional space.

Function header:

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

Return value:

Angle in degrees.

Parameter:

Parameter	Data type	Description
vector_a[]	REAL	Three-dimensional vector a
vector_b[]	REAL	Three-dimensional vector b

Example:

```
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(...)*

Description:

Determination of the minimum and maximum value of a vector.

Function header:

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

Return value:

Minimum value of a vector.

Maximum value of a vector.

Parameter:

Parameter	Data type	Description
vector[]	REAL	One-dimensional vector.
n	INT	Number of elements of the vector.
min_out	REAL	Minimum value of the vector.
max_out	REAL	Maximum value of the vector

Example:

```
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(...)*

Description:

Determination of the average of the elements of a vector.

Function header:

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

Return value:

Average of the elements of a vector.

Parameter:

Parameter	Data type	Description
vector[]	REAL	One-dimensional vector.
n	INT	Number of elements of the vector.
average_out	REAL	Mean of the elements of the vector.

Example:

```
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 (...)**

Description:

Determination of the minimum and maximum value of a vector and its average value.

Function header:

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

Return value:

Minimum value of a vector.

Maximum value of a vector.

Mean of the elements of a vector.

Parameter:

Parameter	Data type	Description
vector[]	REAL	One-dimensional vector.
n	INT	Number of elements of the vector.
average_out	REAL	Mean of the elements of the vector.
min_out	REAL	Minimum value of the vector.
max_out	REAL	Maximum value of the vector.

Example:

```
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 (...)

Description:

Calculation of a vector of length one.

Function header:

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

Return value:

Vector of length one

Parameter:

Parameter	Data type	Description
vector[]	REAL	One-dimensional vector.
n	INT	Number of elements of the vector.

Example:

```
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 Matrices

6.1.9.1 mal_rotMatToRPY (...)

Description:

Calculating RPY angles from a rotation matrix.

Function header:

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

Return value:

Angles A,B,C.

Parameter:

Parameter	Data type	Description
rotMat	REAL[3,3]	rotation matrix.

Example:

```
DECL E6POS baseOrigin, baseX, baseXY
DECL REAL pXdifff[3], pXYdifff[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

pXdifff[1]=baseX.X-baseOrigin.X
pXdifff[2]=baseX.Y-baseOrigin.Y
pXdifff[3]=baseX.Z-baseOrigin.Z
pXYdifff[1]=baseXY.X-baseOrigin.X
pXYdifff[2]=baseXY.Y-baseOrigin.Y
pXYdifff[3]=baseXY.Z-baseOrigin.Z

mal_vectorNormalization(pXdifff[], 3)
mal_vectorNormalization(pXYdifff[], 3)

mal_vectorProduct(pXdifff[], pXYdifff[], pz[])
mal_vectorNormalization(pz[], 3)
mal_vectorProduct(pz[], pXdifff[], py[])

FOR i=1 TO 3
    rotMat[i,1] = pXdifff[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 Geometry

6.1.10.1 *mal_circumference(...)*

Description:

Calculating the circumference of a circle.

Function header:

GLOBAL DEFFCT REAL mal_circumference(radius:IN)

Return value:

Circumference of a circle.

Parameter:

Parameter	Data type	Description
radius	REAL	Circle radius

Example:

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

6.1.10.2 *mal_circularArc(...)*

Description:

Calculation of the circular arc.

Function header:

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

Return value:

Circular arc

Parameter:

Parameter	Data type	Description
radius	REAL	Kreisradius
angle	REAL	Winkel [deg]

Example:

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

```
//circularArc=10.47
```

6.1.11 Random numbers

6.1.11.1 *mal_randLcg(...)*

Description:

Generation of a pseudo-random number. Linear congruence Generator.

Function header:

GLOBAL DEFFCT REAL mal_randLcg(seed:IN)

Return value:

Real value in the range 0.0...1.0.

Parameter:

Parameter	Data type	Description
seed	INT	Start value (seed), e.g. \$ROB_TIMER

Example:

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

6.1.11.2 *mal_randLcgRange(...)*

Description:

Generation of a pseudo-random number in a range. Linear Congruence Generator.

Function header:

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

Return value:

Real value in the range lowerBoundary... upperBoundary.

Parameter:

Parameter	Data type	Description
seed	INT	Start value (seed), e.g. \$ROB_TIMER.
lowerBoundary	REAL	Lower range limit of the random number.
upperBoundary	REAL	Upper range limit of the random number.

Example:

```
INT seed
REAL lowerBoundary, upperBoundary, rand

seed= 123456789
lowerBoundary= 12345
upperBoundary= 987654

rand=mal_randLcgRange(seed, lowerBoundary, upperBoundary)

// rand==162236.844
```

6.2 PhysLib: Physical functions

6.2.1 Temperature

6.2.1.1 *phl_linThermalExpansion(...)*

Description:

Calculation of the linear thermal expansion of a body.

Function header:

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

Return value:

Expansion in the unit [m].

Parameter:

Parameter	Data type	Description
lteCoeff	REAL	Loefficient of expansion [K ⁻¹].
particularLength	REAL	Length [m].
tempDiff	REAL	Temperature difference [K].

Example:

```
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(...)*

Description:

Conversion from Celsius to Kelvin.

Function header:

GLOBAL DEFFCT REAL phl_CelsiusToKelvin(celsius:IN)

Return value:

Temperature in Kelvin.

Parameter:

Parameter	Data type	Description
celsius	REAL	Temperature in Celsius.

Example:

```
REAL tempKelvin, tempCelsius

tempCelsius=22.0
tempKelvin=phl_CelsiusToKelvin(tempCelsius)

// tempKelvin==295.15
```

6.2.1.3 **phl_KelvinToCelsius(...)**

Description:

Conversion from Kelvin to Celsius.

Function header:

GLOBAL DEFFCT REAL phl_KelvinToCelsius(kelvin:IN)

Return value:

Temprature in Celsius

Parameter:

Parameter	Data type	Description
kelvin	REAL	Temperature in Kelvin.

Example:

```
REAL tempKelvin, tempCelsius

tempKelvin=295.15
tempCelsius=phl_KelvinToCelsius(tempKelvin)

// tempCelsius==22.0
```

6.2.1.4 **phl_CelsiusToFahrenheit(...)**

Description:

Celsius to Fahrenheit conversion.

Function header:

GLOBAL DEFFCT REAL phl_CelsiusToFahrenheit(celsius:IN)

Return value:

Temperature in Fahrenheit.

Parameter:

Parameter	Data type	Description
celsius	REAL	Temperature in Celsius.

Example:

```
REAL tempFahrenheit, tempCelsius

tempCelsius=-16.5
tempFahrenheit=phl_CelsiusToFahrenheit(tempCelsius)

// tempFahrenheit==2.3
```

6.2.1.5 *phl_FahrenheitToCelsius(...)*

Description:

Conversion from Fahrenheit to Celsius.

Function header:

GLOBAL DEFFCT REAL phl_FahrenheitToCelsius(fahrenheit:IN)

Return value:

Temperature in Celsius

Parameter:

Parameter	Data type	Description
fahrenheit	REAL	Temperature in Fahrenheit.

Example:

```
REAL tempFahrenheit, tempCelsius

tempFahrenheit=2.3
tempCelsius=phl_FahrenheitToCelsius(tempFahrenheit)

// tempCelsius==-16.5
```

6.2.1.6 *phl_KelvinToFahrenheit(...)*

Description:

Conversion from Kelvin to Fahrenheit.

Function header:

GLOBAL DEFFCT REAL phl_KelvinToFahrenheit(kelvin:IN)

Return value:

Temperature in Fahrenheit

Parameter:

Parameter	Data type	Description
kelvin	REAL	Temperature in Kelvin.

Example:

```
REAL tempFahrenheit, tempKelvin

tempKelvin=315.45
tempFahrenheit=phl_KelvinToFahrenheit(tempKelvin)

// tempFahrenheit==108.14
```

6.2.1.7 *phl_FahrenheitToKelvin(...)*

Description:

Conversion from Fahrenheit to Kelvin.

Function header:

GLOBAL DEFFCT REAL phl_FahrenheitToKelvin(fahrenheit:IN)

Return value:

Temperature in Kelvin

Parameter:

Parameter	Data type	Description
fahrenheit	REAL	Temperature in Fahrenheit.

Example:

```
REAL tempFahrenheit, tempKelvin

tempFahrenheit =108.14
tempKelvin=phl_FahrenheitToKelvin(tempFahrenheit)

// tempKelvin==315.45
```

6.2.2 Time

6.2.2.1 *phl_isLeapYear(...)*

Description:

Leap year check.

Function header:

GLOBAL DEFFCT BOOL phl_isLeapYear(year:IN)

Return value:

- True: Year is leap year.
- False: Year is not a leap year.

Parameter:

Parameter	Data type	Description
year	INT	Year to check for leap years.

Example:

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

6.2.2.2 *phl_daysInMonth(...)*

Description:

Calculation of the number of days in a month.

Function header:

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

Return value:

Number of days in the month.

Parameter:

Parameter	Data type	Description
month	INT	Month in the year [1...12].
year	INT	Year.

Example:


```

INT daysInMonth

daysInMonth=phl_daysInMonth(2, 2000)

// daysInMonth==29

daysInMonth=phl_daysInMonth(3, 2001)

// daysInMonth==31

```

6.2.2.3 *phl_daysInYear(...)*

Description:

Calculation of the number of days in a year.

Function header:

GLOBAL DEFFCT INT phl_daysInYear(year:IN)

Return value:

Number of days in the year.

Parameter:

Parameter	Data type	Description
year	INT	Year.

Example:

```

INT daysInYear

daysInYear=phl_daysInYear(2000)

// daysInYear==366

daysInYear=phl_daysInYear(2001)

// daysInYear==365

```

6.2.2.4 *phl_daysElapsedOfYear(...)*

Description:

Calculation of the number of days in the year up to a date.

Function header:

GLOBAL DEFFCT INT phl_daysElapsedOfYear(actDate:IN)

Return value:

Number of days in the year up to a date.

Parameter:

Parameter	Data type	Description
actDate	DATE	Date.

Example:

```
DECL DATE myDate
INT elapsedDays

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

elapsedDays=phl_daysElapsedOfYear(myDate)

// elapsedDays==282
```

6.2.2.5 ***phl_secInYear(...)***

Description:

Calculation of seconds in a year.

Function header:

GLOBAL DEFFCT INT phl_secInYear(year:IN)

Return value:

Seconds in a year.

Parameter:

Parameter	Data type	Description
year	INT	Year.

Example:

```
INT secInYear

secInYear=phl_secInYear(2000)

// secInYear==31622400

secInYear=phl_secInYear(2001)

// secInYear==31536000
```

6.2.2.6 ***phl_secElapsedOfYear(...)***

Description:

Calculation of the number of seconds in the year up to a date.

Function header:

GLOBAL DEFFCT INT phl_secElapsedOfYear(actDate:IN)

Return value:

Number of seconds in the year up to a date.

Parameter:

Parameter	Data type	Description
actDate	DATE	Date.

Example:

```
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(...)*

Description:

Calculation of the seconds since the year 2000 and a date.

Function header:

GLOBAL DEFFCT INT phl_secSince2000(actDate:IN)

Return value:

Number of seconds

Parameter:

Parameter	Data type	Description
actDate	DATE	Date

Example:

```
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(...)*

Description:

Calculates the time span between two date structures.

Function header:

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

Return value:

Time span as date structure.

Parameter:

Parameter	Data type	Description
date1in	DATE	Date1
date2in	DATE	Date2

Example:

```
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(...)*

Description:

Calculation of the time span in days between two date structures.

Function header:

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

Return value:

Number of days.

Parameter:

Parameter	Data type	Description
date1in	DATE	Date1.
date2in	DATE	Date2.

Example:

```
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(...)*

Description:

Comparison of two DATE structures.

Function header:

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

Return value:

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

Parameter:

Parameter	Data type	Description
date1In	DATE	Date1.
date2In	DATE	Date2.

Example:

```
DECL DATE date1, date2
INT dateCompare

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

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

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

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

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

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

6.2.2.11 *phl_dateStamp(...)*

Description:

Generates a date stamp from a DATE structure.

Function header:

```
GLOBAL DEFFCT INT phl_dateStamp(dateIn:IN)
```

Return value:

Stamp the date in the format [yyyymmdd]

Parameter:

Parameter	Data type	Description
dateIn	DATE	Date.

Example:

```
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(...)*

Description:

Generates a time stamp from a DATE structure.

Function header:

GLOBAL DEFFCT INT phl_timeStamp(dateIn:IN)

Return value:

Stamp of the time in the format [hhmmss]

Parameter:

Parameter	Data type	Description
dateIn	DATE	Date.

```
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: Robot specific features

6.3.1 Transformation

6.3.1.1 **rol_ForwardTransform(...)**

Description:

Calculation of Cartesian coordinates based on axis values.

Function header:

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

Return value:

Cartesian position as result of forward transformation.

Error status (Parameter errState):

- -4: \$TOOL invalid.
- -3: \$BASE invalid.
- -2: Error status invalid.
- -1: Not all axis values initialized.
- 0: Calculation successful.
- 1: Violation of the software limit switch.
- 2: Forward transformation calculation error.

Parameter:

Parameter	Data type	Description
axisValues	E6AXIS	axis values.
softEndCheck	BOOL	Check for violation of software limit switches.
errState	INT	Error status

Example:

```
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(...)**

Description:

Calculation of axis values based on Cartesian coordinates.

Function header:

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

Return value:

Axis values as the result of the inverse transformation.

Error status (Parameter errState):

- -4: \$TOOL invalid.
- -3: \$BASE invalid.
- -2: Error status invalid.
- -1: Not all position values are initialized.
- 0: Calculation successful.
- 1: Violation of the software limit switch.
- 2: Violation of the work space.
- 3: Axis values calculated, software limit switch violated by Turn value.

Parameter:

Parameter	Data type	Description
cartPosition	E6POS	Cartesian position.
startAxis	E6AXIS	Seed Status and Turn if cartPosition does not contain them.
errState	INT	Error status

Example:

```
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(...)**

Description:

Calculation of base coordinates based on three positions.

Function header:

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

Return value:

Frame as base coordinates.

Parameter:

Parameter	Data type	Description
-----------	-----------	-------------

baseOrigin	E6POS	Origin of the coordinate system.
baseX	E6POS	Position on the positive X-axis of the coordinate system.
baseXY	E6POS	Position on the positive XY plane of the coordinate system.

Example:

```
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 Test for equality

6.3.2.1 rol_FrameCompare(...)

Description:

This function determines whether two frame structures lie within a distance and angle limit.

Function header:

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

Return value:

- True: Distance and angle are within the limits..
- False:Distance or angle exceeds the limit.

Parameter:

Parameter	Datentyp	Beschreibung
frame1	FRAME	comparison frame1.
frame2	FRAME	comparison frame2.
distLimit	REAL	Limit value for the distance between the frame structures
angleLimit	REAL	Limit value for the angle of the frame structures

Example:

```
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 Distances

6.3.3.1 *rol_FrameDistance(...)*

Description:

Distance between two FRAME structures.

Function header:

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

Return value:

Distance value in [mm] between frame1 and frame2.

Parameter:

Parameter	Data type	Description
frame1	FRAME	comparison frame1.
frame2	FRAME	comparison frame2.

Example:

```

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(...)*

Description:

Distance between two E6POS structures.

Function header:

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

Return value:

Distance value in [mm] between pos1 and pos2.

Parameter:

Parameter	Data type	Description
pos1	E6POS	comparison position 1.
pos2	E6POS	comparison position 2.

Example:

```
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 Orientations

6.3.4.1 *rol_toolOriToGravity(...)*

Description:

Orientation of the tool coordinate system to the direction of gravity.

Function header:

GLOBAL DEFFCT val_CoordinateAxes_T rol_toolOriToGravity(position:IN)

Return value:

Angular deviations in degrees of the coordinates x,y and z to the direction of gravity.

Parameter:

Parameter	Data type	Description
position	E6POS	Position against which the orientation deviation of the tool coordinate system is calculated.

Example:

```
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(...)*

Description:

Determining whether the movement to a target position is against or with gravity.

Function header:

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

Return value:

Enum:

- upGravity, Movement to the target position is against gravity.
- downGravity, Movement to target position is done with gravity.
- neutralGravity, Movement to the target position is neutral to gravity.

Parameter:

Parameter	Data type	Description
startPos	E6POS	Starting position.
destPos	E6POS	Target position.

Example:

```
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 Timer functions

6.3.5.1 rol_elapsedMsecRobTimer(...)

Description:

Time difference to the current value of \$ROB_TIMER taking into account an overflow.

Function header:

GLOBAL DEFFCT INT rol_elapsedMsecRobTimer(compTime:IN)

Return value:

Time difference between parameter compTime and current time of \$ROB_TIMER in [ms].

Parameter:

Parameter	Data type	Description
compTime	INT	Comparison value for the time difference [ms].

Example:

```
INT startTime, elapsedTime

startTime=$ROB_TIMER
WAIT SEC 1
elapsedTime=rol_elapsedMsecRobTimer(startTime)

//elapsedTime==1000
```

6.3.6 Roboter-Stop

6.3.6.1 rol_robStopAndRelease(...)

Description:

Stops the current movement of the robot and immediately cancels the stop. Continued movement requires operator intervention. The function can only be used in the submit interpreter.

Function header:

GLOBAL DEFFCT BOOL rol_robStopAndRelease(robStopType:IN)

Return value:

- TRUE – A stop initiated by rol_robStop(...) has been removed.
- FALSE – No stop was previously initiated by rol_robStop(...).

Parameter:

Parameter	Data type	Description
robStopType	ROB_STOP_T	- #Ramp_Down (ramp stop) - #Path_Maintaining (path-maintaining stop)

6.3.6.2 rol_robStop(...)

Description:

Stops the current movement of the robot. A corresponding status message is sent. The function can only be used in the submit interpreter.

Function header:

GLOBAL DEFFCT BOOL rol_robStop(robStopType:IN)

Return value:

- TRUE – Stop runs successfully.
- FALSE – Bad parameter.

Parameter:

Parameter	Data type	Description
robStopType	ROB_STOP_T	- #Ramp_Down (ramp stop) - #Path_Maintaining (path-maintaining stop)

6.3.6.3 **rol_robStopRelease()**

Description:

A stop initiated by rol_robStop() is canceled. The function can only be used in the submit interpreter.

Function header:

GLOBAL DEFFCT BOOL rol_robStopRelease()

Return value:

- TRUE – A stop initiated by rol_robStop(...) has been removed.
- FALSE – No stop was previously initiated by rol_robStop(...).

Parameter:

No parameters.

6.3.7 Trace

6.3.7.1 **rol_TraceStart(...)**

Description:

Start a trace recording.

Function header:

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

Parameter:

Parameter	Data type	Description
traceConfig[]	CHAR	Trace configuration name.
traceName[]	CHAR	Trace recording name. If the parameter is not initialized, the base name from the configuration is used as the recording name.

Example:

```

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()**

Description:

Stopping a running trace recording.

Function header:

GLOBAL DEF rol_TraceEnd()

Parameter:

No parameter

Example:

please refer „rol_TraceStart(..)“

6.4 AxisLib: Axis-specific functions

6.4.1 Positions

6.4.1.1 **axl_axisCmdPos(...)**

Description:

Commanded position of an axis.

Function header:

GLOBAL DEFFCT REAL axl_axisCmdPos(axNo:IN)

Return value:

Current commanded position.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```

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(...)

Description:

Actual position of an axis.

Function header:

GLOBAL DEFFCT REAL axl_axisMeasPos(axNo:IN)

Return value:

Current actual position.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```

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(...)

Description:

Current motor angle.

Function header:

GLOBAL DEFFCT REAL axl_axisMot(axNo:IN)

Return value:

Current motor angle in degrees.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```
REAL motPos
INT axNo

axNo=7
motPos=axl_axisMot(axNo)

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

6.4.1.4 axl_axisPosInfo(...)

Description:

Information about the positions of an axis.

Function header:

GLOBAL DEFFCT axl_AxisPosInfo_T axl_axisPosInfo(axNo:IN)

Return value:

Structure of type axl_AxisInfo_T with the following components:

- posCmd: commanded position [deg] / [mm]
- posAct: actual position [deg] / [mm]
- posLag: position lag [deg] / [mm]
- posMot: motor angle of the axis [deg]

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```
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(...)

Description:

Reading the position of an additional axis in an E6POS structure.

Function header:

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

Return value:

Position of an additional axis.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)
e6position	E6POS	E6POS - structure

Example:

```
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(...)

Description:

Setting the position of an additional axis in an E6POS structure.

Function header:

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

Return value:

E6POS structure with value of exaxPos occupied structure element of an additional axis.

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6).
exaxPos	REAL	Position specification of an additional axis.
posStrucIn	E6POS	E6POS structure whose structure element is assigned for an additional

		axis.
--	--	-------

Example:

```
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(...)

Description:

Setting the position of an axis in an E6AXIS structure.

Function header:

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

Return value:

E6AXIS structure with value of axPos occupied structure element of an axis.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12).
axPos	REAL	Position specification of an axis.
axisStrucIn	E6AXIS	E6AXIS structure whose structure element is assigned for an additional axis.

Example:

```
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(...)

Description:

Reading the position of an axis in an E6AXIS structure.

Function header:

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

Return value:

Position of an axis.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12).
axisStrucIn	E6AXIS	E6AXIS structure whose structure element is assigned for an additional axis.

Example:

```
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(...)

Description:

Writing an axis position in an E6AXIS structure.

Function header:

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

Return value:

E6AXIS - structure with value of axPos occupied structure element. Components of the other axes are not initialized.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12).

axPos	REAL	Position specification of an axis.
-------	------	------------------------------------

Example:

```
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 Distances

6.4.2.1 axl_axisDistPosToPos(...)

Description:

Axis differences of two E6POS structures.

Function header:

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

Return value:

Axis distances as E6AXIS structure between pos1 and pos2.

Parameter:

Parameter	Data type	Description
startAxis	E6AXIS	Seed value Status and Turn if pos1 does not contain them.
pos1	E6POS	comparison position 1
pos2	E6POS	comparison position 2

Example:

```
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(...)

Description:

Differences of the axes from the current position to a target position.

Function header:

GLOBAL DEFFCT E6AXIS axl_e6AxisDistToDestPos(destPos:IN)

Return value:

Distance of all axes.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
destPos	E6POS	Zielposition

Example:

```
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(...)

Description:

Difference of an additional axis from the current position to a target position.

Function header:

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

Return value:

Distance of an additional axis.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6).
destPos	E6POS	Target position.

Example:

```
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(...)

Description:

Axis differences of two E6AXIS structures.

Function header:

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

Return value:

Difference of axes of two E6AXIS structures.

Unit [mm] for a linear axis.

Unit [deg] for a rotary axis.

Parameter:

Parameter	Data type	Description
startAxis	E6AXIS	Axis position 1
destAxis	E6AXIS	Axis position 2

Example:

```
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(...)**

Description:

Axis differences of the basic axes of two E6AXIS structures.

Function header:

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

Return value:

Difference of the basic axes of two AXIS structures.

Parameter:

Parameter	Data type	Description
startAxis	AXIS	basic axis position 1
destAxis	AXIS	basic axis position 2

Example:

```
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 **Velocities**

6.4.3.1 **axl_maxVelOutput(...)**

Description:

Maximum output speed of an axis.

Function header:

GLOBAL DEFFCT REAL axl_maxVelOutput(axNo:IN)

Return value:

Output-side maximum speed of an axis.

Unit [deg/s] for a rotary axis.

Unit [mm/s] for a linear axis.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```

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(...)

Description:

Conversion of an absolute speed into a relative speed.

Function header:

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

Return value:

Relative speed (1..100%) of an axis in relation to the maximum speed.

Unit [%]

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12).
absVel	REAL	Absolute speed [mm/s] / [deg/s].

Example:

```

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(...)

Description:

Conversion of a relative speed into an absolute value.

Function header:

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

Return value:

Output-side absolute speed of an axis.

Unit [deg/s] for a rotary axis.

Unit [mm/s] for a linear axis.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)
relVel	REAL	Relative speed of an axis in [%]

Example:

```
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(...)

Description:

Information about the speeds of an axis.

Function header:

GLOBAL DEFFCT axl_AxisVelInfo_T axl_axisVelInfo(axNo:IN)

Return value:

Structure of type axl_AxisVelInfo_T with the following components:

- velCmd: commanded output speed [deg/s] / [mm/s]
- velAct: output-side actual speed [deg/s] / [mm/s]
- velLag: position following error on the output side [deg/s] / [mm/s]

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```
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 Torques

6.4.4.1 *axl_axisTorqueInfo(...)*

Description:

Information about the torques of an axis.

Function header:

GLOBAL DEFFCT axl_AxisTorqueInfo_T axl_axisTorqueInfo(axNo:IN)

Return value:

Structure of type axl_TorqueInfo_T with the following components:

- tqAct: Current engine torque in [Nm] / [N].
- tqMax0: Maximum continuous engine torque in [Nm] / [N].
- tqMax: Absolute maximum engine torque in [Nm] / [N].
- tqHold: Holding torque at current axis position in [Nm] / [N].

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12).

Example:

```
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 Transmission

6.4.5.1 *axl_linRatioSpindleDrive(...)*

Description:

Conversion of the pitch of a spindle gear into the transmission ratio for linear axes.

Function header:

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

Return value:

Linear ration.

Unit: [rev/mm].

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)
spindlePitch	REAL	Spindle pitch [mm/U]

Example:

```
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 Axis state

6.4.6.1 axl_axisMastered(...)

Description:

Check whether there is a valid mastering for an axis.

Function header:

GLOBAL DEFFCT BOOL axl_axisMastered(axNo:IN)

Return value:

State of the axis adjustment:

- True: Axis mastered.
- False: Axis not mastered.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```

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(...)**

Description:

Check whether motor brake of an axis is open.

Function header:

GLOBAL DEFFCT BOOL axl_brakeOpen(axNo:IN)

Return value:

State of the brake of an axis.

- True: brake open.
- False: brake closed.

Parameter:

Parameter	Data type	Description
axNo	INT	Axis number (1...12)

Example:

```

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_exAxisAsynchron(...)**

Description:

Check whether an additional axis is in the asynchronous operating state.

Function header:

GLOBAL DEFFCT BOOL axl_exAxisAsynchron(exaxNo :IN)

Return value:

Synchronous / asynchronous operating status of the additional axis:

- True: Axis asynchronous.

- False: Axis synchronous.

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)

Example:

```

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(...)**

Description:

Switch additional axis to synchronous mode.

Function header:

GLOBAL DEF axl_exAxSynchron(exaxNo:IN)

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)

Example:

```

INT exaxNo

exaxNo=2
axl_exAxSynchron(exaxNo)

```

6.4.6.5 **axl_exAxAsynchron(...)**

Description:

Switch additional axis to asynchronous operation.

Function header:

GLOBAL DEF axl_exAxAsynchron(exaxNo:IN)

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)

Example:

```
INT exaxNo

exaxNo=2
axl_exAxAsynchron(exaxNo)

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

6.4.6.6 axl_exAxIsCoupled(...)

Description:

Check whether an additional axle is coupled.

Function header:

GLOBAL DEFFCT BOOL axl_exAxIsCoupled(exaxNo :IN)

Return value:

Status of the coupling of an additional axis:

- True: Axis coupled.
- False: Axis decoupled

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)

Example:

```
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(...)

Description:

Couple additional axis.

Coupling and decoupling requires a corresponding configuration.

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

Attribute: Coupling

Function header:

GLOBAL DEF axl_exAxCouple(exaxNo:IN)

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)

Example:

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

6.4.6.8 axl_exAxDecouple(...)

Description:

Decouple the additional axis.

Coupling and decoupling requires a corresponding configuration.

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

Attribute: Coupling

Function header:

GLOBAL DEF axl_exAxDecouple(exaxNo:IN)

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6)

Example:

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

6.4.7 Movements

6.4.7.1 axl_exAxAsyncMovement(...)

Description:

Asynchronous motion command for an additional axis.

Function header:

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

Parameter:

Parameter	Data type	Description
exaxNo	INT	Number of the additional axis (1...6).
destPos	REAL	Target position.

Example:

```
INT exaxNo
REAL destPos

exaxNo=2
destPos=120.0

axl_exAxAsyncMovement(exaxNo, destPos)
```

6.5 MsgLib: User messages

6.5.1 Fire messages

6.5.1.1 *msl_FireMsg()*

Description:

Submission of a message with the option of passing integer or real parameters. The sum of the parameters is limited to three.

Function header:

```
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	Data type	Description
msgTyp	Enum	Message type • #notify: Notify message. • #state: Status message. • #quit: Quit message. • #waiting: Waiting message.
msgModule[]	CHAR	Sender or name of the kxr message database • max. character length 26.
msgNo	INT	Message number.
msgTxt[]	CHAR	Message tex: • max. character length: 80

krlMsgOpt	Struktur	Behavior with message output. Structure of type KrlMsgOpt_T: • vl_stop ◦ TRUE: Message output stops advance. ◦ FALSE: No advance stop for message output. • clear_p_reset ◦ TRUE: Messages are deleted when the program is cancelled or reset. ◦ FALSE: Messages are not deleted when the program is cancelled or reset. • clear_p_SAW ◦ TRUE: Messages are deleted when the block is selected. ◦ FALSE: Messages are not deleted when a block is selected. • log_to_DB ◦ TRUE: Messages are logged in the event log. ◦ FALSE: Messages are not logged.
msgHandle	INT	Return value: • -1: The message could not be sent. • 0: Notify message. • >0: The message was sent successfully.
iPar[1..3]	INT	Integer parameters 1...3.
rPar[1..3]	REAL	Real parameters 1...3.

Example:

;Notify message

```
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()

Description:

Submission of a dialog message with the option of transferring integer or real parameters. The sum of the parameters is limited to three.

Function header:

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	Data type	Description
msgModule[]	CHAR	Sender or name of the kxr message database: max. character length 26
msgNo	INT	Message number.
msgTxt[]	CHAR	Message text: max. character length: 80
krlMsgOpt	Struktur	Behavior with message output. Structure of type KrlMsgOpt_T: - vl_stop - TRUE: Message output stops advance. - FALSE: No advance stop for message output. - clear_p_reset - TRUE: Messages are deleted when the program is cancelled or reset. - FALSE: Messages are not deleted when the program is cancelled or reset. - clear_p_SAW - TRUE: Messages are deleted when the block is selected. - FALSE: Messages are not deleted when a block is selected. - log_to_DB - TRUE: Logging of messages in the event log. - FALSE: No logging of the messages.

msgHandle	INT	Return value: • -1: The message could not be sent. • >0: The message was sent successfully.
keySelected	INT	Return value of the softkey selected by the user.
KeyTxt[1...7]	CHAR	Identifier of the softkeys: • max. character length: 10.
IPar[1..3]	INT	Integer parameters.
RPar[1..3]	REAL	Real parameters.

Example:

```
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()*

Description:

Dispatch of a dialog message with the possibility of an acknowledgment via inputs. An input can be assigned to each softkey.

Integer or real parameters can be passed to the message text. The number of parameters is limited to three.

Function header:

```
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	Data type	Description
msgModule[]	CHAR	Sender or name of the kxr message database: • max. character length 26.
msgNo	INT	Message number.
msgTxt[]	CHAR	Message text:

		max. character length: 80.
krlMsgOpt	Struktur	Behavior with message output. Structure of type KrlMsgOpt_T: - vl_stop - TRUE: Message output stops advance. - FALSE: No advance stop for message output. - clear_p_reset - TRUE: Messages are deleted when the program is cancelled or reset. - FALSE: Messages are not deleted when the program is cancelled or reset. - clear_p_SAW - TRUE: Messages are deleted when the block is selected. - FALSE: Messages are not deleted when a block is selected. - log_to_DB - TRUE: Messages are logged in the event log. - FALSE: Messages are not logged.
msgHandle	INT	Return value: -1: The message could not be sent. >0: The message was sent successfully.
keySelected	INT	Return value of the softkey selected by the user or the corresponding input.
keyTxt[1...7]	CHAR	Identifier of the softkeys: max. character length: 10
iPar[1..3]	INT	Integer parameters.
rPar[1..3]	REAL	Real parameters.
di_QuitExt[]	INT	Inputs for dialog acknowledgment.

Example:

```
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()*

Description:

Sending a message with the option of passing character strings as parameters.

Function header:

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

Parameter:

Parameter	Data type	Description
msgTyp	Enum	Message type • #notify: Notify message. • #state: Status message. • #quit: Status message. • #waiting: Wartemeldung.
msgModule[]	CHAR	Sender or name of the kxr message database: • max. character length 26.
msgNo	INT	Message number.
msgTxt[]	CHAR	Message text: • max. character length: 80.
krlMsgOpt	Struktur	Behavior with message output. Structure of type KrlMsgOpt_T: • vl_stop ◦ TRUE: Message output stops advance. ◦ FALSE: No advance stop for message output. • clear_p_reset ◦ TRUE: Messages are deleted when the program is cancelled or reset. ◦ FALSE: Messages are not deleted when the program is cancelled or reset. • clear_p_SAW ◦ TRUE: Messages are deleted when the block is selected.

		◦ FALSE: Messages are not deleted when a block is selected. • log_to_DB ◦ TRUE: Messages are logged in the event log. ◦ FALSE: Messages are not logged.
msgHandle	INT	Return value: • -1: The message could not be sent. • 0: Notify message. • >0: The message was sent successfully.
strPar[1..3]	CHAR	String parameters: • max. character length 26

Example:

```
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()

Description:

Submission of a message with the option of passing a frame as a parameter.

Function header:

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

Parameter:

Parameter	Data type	Description
msgTyp	Enum	Message type: • #notify: Notify message • #state: Status message • #quit: Quittiermeldung • #waiting: Wartemeldung
msgModule[]	CHAR	Sender or name of the kxr message database: • max. character length 26.

msgNo	INT	Message number.
msgTxt[]	CHAR	Message text: • max. character length: 80.
krlMsgOpt	Struktur	Behavior with message output. Structure of type KrlMsgOpt_T: • vl_stop ◦ TRUE: Message output stops advance. ◦ FALSE: No advance stop for message output. • clear_p_reset ◦ TRUE: Messages are deleted when the program is cancelled or reset. ◦ FALSE: Messages are not deleted when the program is cancelled or reset. • clear_p_SAW ◦ TRUE: Messages are deleted when the block is selected. ◦ FALSE: Messages are not deleted when a block is selected. • log_to_DB ◦ TRUE: Messages are logged in the event log. ◦ FALSE: Messages are not logged.
msgHandle	INT	Return value: • -1: The message could not be sent. • 0: Notify message. • >0: The message was sent successfully.
frameData	Frame	Frame-Parameter • The number of initialized components of the frame structure is arbitrary.

Example:

```
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 Message status

6.5.2.1 *msl_ExistsMsg(...)*

Description:

Checks whether a message is pending.

Function header:

GLOBAL DEFFCT BOOL msl_ExistsMsg(msgHandle:IN)

Return value:

- TRUE – Notification is currently pending.
- FALSE –message is not pending.

Parameter:

Parameter	Data type	Description
msgHandle	INT	Handle of the message to be checked.

Example:

```
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(...)

Description:

Checks whether a dialog message is pending.

Function header:

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

Return value::

- TRUE – Dialog message existst.
- FALSE – Dialog message not exists.

Parameter:

Parameter	Data type	Description
msgHandle	INT	Handle of the dialog message to be checked.
keySelected	INT	Button number • 1...7: Button with which the user has confirmed the dialog. • 0: Dialog was ended in the program and not by the user.

6.5.2.3 msl_Buffer(..)

Description:

Reading the message buffer and the number of messages in the buffer.

Function header:

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

Parameter:

Parameter	Data type	Description
msgBuf[]	MsgBuf_T	Message buffer: • type: Message type (#sys_quit, #usr_State, ...). • nr: Message number. • Modul[]. • msg_txt[]. • par_type. • par_txt. • handle. The buffer contains no notification messages.
msgNo	INT	Number of messages in the buffer.

Example:

```
INT msgHandle
BOOL bReturn

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 'B0010011100011100'}
```

6.5.3 Clear messages

6.5.3.1 msl_Clear(..)

Description:

Delete user messages. Notify messages (msgHandle==0) cannot be deleted.

Function header:

GLOBAL DEFFCT BOOL msl_Clear(msgHandle:IN)

Return value:

- TRUE – Delete message(s) successful.

- FALSE – Deletion of message(s) failed.

Parameter:

Parameter	Data type	Description
msgHandle	INT	Handle of the message to be deleted: • -1: All messages of the process are deleted • -99: Messages from all processes are deleted.

Example:

```

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 Parameter assignment

6.5.4.1 *msl_paramInt(..)*

Description:

Assign an integer value to the placeholder.

Function header:

GLOBAL DEFFCT KrlMsgPar_T msl_paramInt(paramInt:IN)

Return value:

Structure of type KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_INT=paramInt.

Parameter:

Parameter	Data type	Description
paramInt	INT	Integer value replacing a placeholder.

6.5.4.2 *msl_paramReal(..)*

Description:

Assign a real value to the placeholder.

Function header:

GLOBAL DEFFCT KrlMsgPar_T msl_paramReal(paramReal:IN)

Return value:

Structure of type KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_REAL=paramReal.

Parameter:

Parameter	Data type	Description
paramReal	REAL	Real value replacing a placeholder.

6.5.4.3 *msl_paramBool(..)*

Description:

Assign a boolean value to the placeholder.

Function header:

GLOBAL DEFFCT KrlMsgPar_T msl_paramBool(paramBool:IN)

Return value:

Structure of type KrlMsgPar_T:

- PAR_TYPE=#VALUE.
- PAR_BOOL=paramBool.

Parameter:

Parameter	Data type	Description
paramBool	BOOL	Boolean value that replaces a placeholder.

6.5.4.4 *msl_paramString(..)*

Description:

Assign a string to the placeholder.

Function header:

GLOBAL DEFFCT KrlMsgPar_T msl_paramString(paramString[:IN])

Return value:

Structure of type KrlMsgPar_T:

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

Parameter:

Parameter	Data type	Description
-----------	-----------	-------------

paramString	CHAR	String replacing a placeholder.
-------------	------	---------------------------------

6.6 FileLib: Handling of Files

6.6.1 Write data to disk

Writing data or text to files on disk. Each entry is numbered consecutively and receives a time stamp.

6.6.1.1 *fil_LogValuesToCsv(...)*

Description:

Writing data in csv format to HDD/SSD. The file is written to the "C:\KRC\ROBOTER\UserFiles" directory.

The character length of a line is limited to 1024 characters. A maximum of 10 files with 1000 lines each are written. The files are then overwritten in a ring buffer.

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

The csv files are saved in a KRCdiag.

Function header:

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	Data type	Description
fileName[]	CHAR	Filename.
headLine[]	CHAR	Header. Maximum length is 462 characters.
fileCount	fil_csvFileCount_T	The structure contains file number and line number components. The components are incremented within the routine.
rPar[1...10]	REAL	Parameters that are written.

Example:

```
;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(...)***

Description:

Writing line to a csv file on HDD/SSD. The file is written to the "C:\KRC\ROBOTER\UserFiles" directory.

The character length of a line is limited to 1024 characters. A maximum of 10 files with 1000 lines each are written. The files are then overwritten in a ring buffer.

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

The csv files are saved in a KRCdiag.

Function header:

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

Parameter:

Parameter	Data type	Description
fileName[]	CHAR	Filename.
headLine[]	CHAR	Header. Maximum length is 462 characters.
fileCount	fil_csvFileCount_T	The structure contains file number and line number components. The components are incremented within the routine.
logLine[]	CHAR	Line written to file. The line length is limited to 438 characters.

Example:

```
;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(...)*

Description:

Writing data in txt format to HDD/SSD.

Functionally identical to *fil_LogLineToCsv*.

6.6.1.4 *fil_LogMessage(...)*

Description:

Writing character strings to a *.log file on HDD/SSD. The file is created in the "C:\KRC\ROBOTER\UserFiles" directory. 2 integer and 8 real parameters can be transferred to a line. Placeholders are to be provided in the text for the parameters.

The character length of a line is limited to 1024 characters. A maximum of 10 files with 1000 lines each are written. The files are then overwritten in a ring buffer.

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

Function header:

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	Data type	Description
logLine[]	CHAR	String to write to a file. If parameters are passed, placeholders must be provided (%1, %2,...).
iPar[1..2]	INT	Parameter.
rPar[3..10]	REAL	Parameter.

Example:

```
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 File handling

6.6.2.1 *fil_FileExists(...)*

Description:

Check if a file exists.

The file is searched for in the "C:\KRC\ROBOTER\UserFiles" directory.

Function header:

GLOBAL DEFFCT BOOL fil_FileExists(fileName[:IN])

Return value:

- TRUE – File exists.
- FALSE –File not exists

Parameter:

Parameter	Data type	Description
fileName[]	CHAR	Filename

Example:

```
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(...)*

Description:

Deleting a file on HDD.

The file is searched for in the "C:\KRC\ROBOTER\UserFiles" directory and deleted if it exists.

Function header:

GLOBAL DEFFCT BOOL fil_FileRemove(fileName[:IN])

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
fileName[]	CHAR	Filename.

Example:

```
BOOL bResult

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

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

//bResult==TRUE
```

6.7 StringLib: Handling of character strings

6.7.1 Convert variable values

6.7.1.1 *stl_boolToString(..)*

Description:

Converts a boolean value into a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
valueBool	BOOL	Value to be converted to a string.
valueStr[]	CHAR	Value of valueBool as a string

Example:

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

bVariable=TRUE

bSuccess=stl_boolToString(bVariable, boolString[])

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

6.7.1.2 stl_intToString(..)

Description:

Converts an integer value to a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
valueInt	INT	Integer value converted to a string.
valueStr[]	CHAR	Value of the integer as a string.

Example:

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

iNum=12345

bSuccess=stl_intToString(iNum, intString[])

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

6.7.1.3 stl_realToString(..)

Description:

Converts a real value to a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
valueReal	REAL	Value to be converted to a string.
valueStr[]	CHAR	Value of valueReal as a string. The value is rounded to two decimal places. Exponential notation is used for values ≥ 100000.0 or ≤ -100000.0 .

Example:

```
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(..)

Description:

Converts a frame structure into a character string.

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

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
valueFrame	Frame	Value to be converted to a string. Uninitialized components are pre-assigned with "0".
valueStr[]	CHAR	Value of valueFrame as a string.

Example:

```
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(..)*

Description:

Converts an E6AXIS structure to a character string.

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

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
valueAxis	E6AXIS	Structure that is converted to a string. Uninitialized components are pre-assigned with "0".
valueStr[]	CHAR	Value of valueAxis as a string.

Example:

```

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(..)**

Description:

Converts the date of a date structure into a character string.

Syntax: „YYYY-MM-DD“

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
dateStruc	DATE	Structure of type date.
dateTimeString[]	CHAR	Date as a string. Character length >=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(..)**

Description:

Converts the time of a date structure into a character string.

Syntax: „HH.MM.SS.“

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
dateStruc	DATE	Structure of type date.
timeString[]	CHAR	Time as a string. Character length >=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(..)

Description:

Converts a date structure into a character string according to the ISO-8601 standard.

Syntax: „YYYY-MM-DDTHH.MM.SS.“

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
dateStruc	DATE	Structure of type date converted to a string.
dateTimeString []	CHAR	Value of dateStruc as a string. Character length >=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(..)**

Description:

Converts an ASCII string to an integer.

Function header:

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

Return value:

- TRUE - Conversion of ASCII string to integer successful.
- FALSE - Conversion of ASCII string to integer failed.

Parameter:

Parameter	Data type	Description
valueStr[]	CHAR	ASCII character string consisting of signs and digits.
valueInt	INT	Signed integer.

Example:

```
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(...)**

Description:

Converts the value of an integer into a character string. The character string is padded with left-justified zeros.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
iValue	INT	Integer value converted to a string.
nullValString	CHAR	Integer value as a string with left-justified zero padding.

Example:

```
DECL CHAR padString[20]
BOOL bResult

bResult=stl_padLeadingZeros(123456, padString[])
//padString[] == "000000000000000123456"

bResult=stl_padLeadingZeros(-56, padString[])
//padString[] == "0000000000000000-56"
```

6.7.2 Edit strings

6.7.2.1 stl_toUpper(..)

Description:

Converting a string to uppercase.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	String to be converted to uppercase.
lineUpper[]	CHAR	Uppercase string. Declaration of the length of lineUpper[] >= the length of line[].

Example:

```
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(..)**

Description:

Converting a string to lowercase.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	String to be converted to lowercase.
lineLower[]	CHAR	Lowercase string. Declaration of length of lineLower[] >= length of line[].

Example:

```
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 (..)**

Description:

Replacing a placeholder in a character string. '%s' is used as a placeholder. Only one wildcard is allowed.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
lineMerge[]	CHAR	Character string with replaced placeholder.
lineParam[]	CHAR	Character string with placeholder. The length may not exceed 470 characters.

strReplace	CHAR	String to be inserted.
------------	------	------------------------

Example:

```
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(..)

Description:

Replacing placeholders in a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
lineMerge[]	CHAR	String with replaced placeholders.
lineParam[]	CHAR	Character string with placeholder.
paramList[]	fil_msgLogParam_T	parameter list.

Example:

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

;merging int-, real-, Bool paramater 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 paramater=%3")

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

//lineMerge[]=="Integer parameter=123, Real parameter= 456.79, Bool paramater=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(..)**

Description:

Removal of spaces at the beginning or at the end of a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	Character string whose leading and trailing spaces are removed.
lineTrim[]	CHAR	Character string without spaces at the beginning or end.

Example:

```

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(..)**

Description:

Removal of spaces at the beginning of a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	Character string whose leading spaces are removed.
lineTrim[]	CHAR	Character string without leading spaces.

Example:

```
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(..)**

Description:

Removal of spaces at the end of a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	Character string whose leading and trailing spaces are removed.
lineTrim[]	CHAR	Character string without spaces at the beginning or end.

Example:

```
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(..)**

Description:

Returns part of a string. Starting from a starting position with a certain length.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	Character string from which a part is extracted.
startAt	INT	Start position (>0...<=length line[]).
length	INT	Length of substring (>0...<=length line[]).
subString[]	CHAR	Substring.

Example:

```
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(..)**

Description:

Separates a string into a maximum of 10 substrings using a separator.

Function header:

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)

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	String for division. The maximum allowed length is 470 characters.
startAt	INT	Start position (>0...<=length line[])
separator	CHAR	Delimiter.
numSplit	INT	Number of determined substrings.
SplitString1[1..10][]	CHAR	Substrings.

Example:

```
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(..)

Description:

Searching for a text within a character string.

Function header:

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

Return value:

- TRUE – Operation successful.
- FALSE – Operation failed.

Parameter:

Parameter	Data type	Description
line[]	CHAR	String to be searched.

find[]	CHAR	Text fragment to be searched for.
startAt	INT	Position from which to search (1...length of character string).
caseSens	ENUM	Case sensitive: • #CASE_SENS: Upper and lower case is taken into account. • #NOT_CASE_SENS: Upper and lower case is not taken into account.
numHits	INT	Number of text fragments found.
indexOfList[]	INT	List with the positions of the found text fragments.

Example:

```

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 Check strings

6.7.3.1 *stl_charIsNumeric* (..)

Description:

Checking whether a character represents a digit.

Function header:

GLOBAL DEFFCT BOOL stl_charIsNumeric(testChar:IN)

Return value:

- TRUE – The character represents a digit.
- FALSE – The character does not represent a digit.

Parameter:

Parameter	Data type	Description
testChar	CHAR	Sign for examination.

Example:

```
DECL CHAR myChar
BOOL bReturn

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

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

6.7.3.2 **stl_charIsAlphaNumeric (.)**

Description:

Check whether a character represents a letter or a digit.

Function header:

GLOBAL DEFFCT BOOL stl_charIsAlphaNumeric(testChar:IN)

Return value:

- TRUE – The character represents a number or a letter.
- FALSE – The character does not represent a digit or a letter.

Parameter:

Parameter	Data type	Description
testChar	CHAR	Sign for examination.

Example:

```
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 (..)**

Description:

Check whether a character represents a lowercase letter.

Function header:

GLOBAL DEFFCT BOOL stl_charIsLowerCase(testChar:IN)

Return value:

- TRUE – The character is a lowercase letter.
- FALSE – The character is not a letter or a capital letter.

Parameter:

Parameter	Data type	Description
testChar	CHAR	Sign for examination.

Example:

```
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 (..)**

Description:

Check if a character represents an uppercase letter.

Function header:

GLOBAL DEFFCT BOOL stl_charIsUpperCase(testChar:IN)

Return value:

- TRUE – The character is a capital letter.
- FALSE – The character is not a letter or a lower case letter.

Parameter:

Parameter	Data type	Description
testChar	CHAR	Sign for examination.

Example:

```

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 (...)**

Description:

Check whether a character string is empty (initialized length=0) or consists only of spaces.

Function header:

GLOBAL DEFFCT BOOL stl_stringIsNullOrSpace(testString[:OUT)

Return value:

- TRUE – The string is empty or consists entirely of spaces.
- FALSE – The character string is not empty and does not consist exclusively of spaces.

Parameter:

Parameter	Data type	Description
testString[]	CHAR	Sign for examination.

Example:

```

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(...)**

Description:

Check whether a character string contains a valid IP address (V4).

Function header:

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

Return value:

- TRUE - The string represents a valid IP address.
- FALSE - The string does not represent a valid IP address.

Parameter:

Parameter	Data type	Description
ipAdress[]	CHAR	Sign for examination.
iOctet[]	INT	Field for 4 octets. If the character string does not represent a valid IP address, the octets are filled with -1.

Example:

```
;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: Variables and Types

6.8.1 Type conversion

6.8.1.1 *val_UIntToInt(...)*

Description:

Conversion of an unsigned integer to a signed integer. Variable data lengths of the unsigned integer (signal agreement) can be viewed with the "bitSize" parameter. Signal declarations with data lengths less than 32 can be converted into a signed number on the basis of two's complement.

Function header:

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

Return value:
Signed integer.

Parameter:

Parameter	Data type	Description
uintValue	INT	Unsigned integer.
bitSize	INT	Bit length of the unsigned integer (2...31).

Example:

```
INT iResult, bitSize, uintValue

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

//iResult=-3
```

6.8.1.2 **val_IntToUInt(...)**

Description:

Conversion of a signed integer to an unsigned integer. Variable data lengths of the signed integer (signal agreement) can be viewed with the "bitSize" parameter. The conversion is based on two's complement.

Function header:

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

Return value:

Vorzeichenlose Ganzzahl.

Parameter:

Parameter	Data type	Description
intValue	INT	Signed integer (-1073741824 ...1073741823).
bitSize	INT	Bit length of the signed integer (2...31).

Example:

```
INT iResult, bitSize, intValue

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

//iResult=13
```

6.8.1.3 **val_bytesToInt(...)**

Description:

Convert a four element byte array to an integer.

Function header:

DEFFCT INT val_bytesToInt(Bytes[:OUT])

Return value:

Signed integer.

Parameter:

Parameter	Data type	Description
Bytes[]	CHAR	Byte array of size 4.

Example:

```
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(...)

Description:

Bit shift to the right.

Function header:

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

Return value:

Bit shift result.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer (1...maxInt).
bitsNo	INT	Number of bits to shift (>0).

6.8.2 Bit-conversion

6.8.2.1 val_bitConvIntToBuffer(...)

Description:

Conversion of an integer to a four-element byte array.

Function header:

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

Return value:

Byte array of data type CHAR with field size 4.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer to convert to a byte array.

6.8.2.2 val_bitConvRealToBuffer(...)

Description:

Conversion of a real to a four-element byte array.

Function header:

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

Return value:

Byte array from data type CHAR with field size 4.

Parameter:

Parameter	Data type	Description
realValue	REAL	Real number to convert to a byte array.

6.8.2.3 val_bitConvBoolToBuffer(...)

Description:

Conversion of a boolean parameter to a byte.

Function header:

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

Return value:

Byte array of data type CHAR with field size 1.

Parameter:

Parameter	Data type	Description
boolValue	BOOL	Data type bool to convert to a byte array

6.8.2.4 **val_bitConvBufferToInt(...)**

Description:

Convert a four element byte array to an integer.

Function header:

GLOBAL DEFFCT INT val_bitConvBufferToInt(intBuff[:IN])

Return value:

Integer.

Parameter:

Parameter	Data type	Description
intBuff[]	CHAR	Byte array to convert to an integer.

6.8.2.5 **val_bitConvBufferToReal(...)**

Description:

Conversion of a four-element byte array to a real number.

Function header:

GLOBAL DEFFCT INT val_bitConvBufferToReal(realBuff[:IN])

Return value:

Real number.

Parameter:

Parameter	Data type	Description
realBuff[]	CHAR	Byte array to convert to a real number.

6.8.2.6 **val_bitConvBufferToBool(...)**

Description:

Conversion of a one-element byte array to a boolean variable.

Function header:

GLOBAL DEFFCT BOOL val_bitConvBufferToBool(boolBuff[:IN])

Return value:

Boolean variable.

Parameter:

Parameter	Data type	Description
boolBuff[]	CHAR	Byte array to convert to a boolean variable.

6.8.2.7 *val_bitConvRealToInt(...)*

Description:

Conversion of a real number to a byte array and from a byte array to an integer.

Function header:

GLOBAL DEFFCT INT val_bitConvRealToInt(realValue:IN)

Return value:

Integer converted from a real number.

Parameter:

Parameter	Data type	Description
realValue	REAL	Real number to convert to an integer.

6.8.2.8 *val_bitConvIntToReal(...)*

Description:

Conversion of an integer to a byte array and from a byte array to a real number.

Not all integers can be converted to a real number. In the event of an error, the message "realValue= NaN value invalid" is output. The conversion of an integer that was previously determined using the val_bitConvRealToInt() function (exchanging a REAL variable using a signal declaration) always results in a valid return value.

Function header:

GLOBAL DEFFCT REAL val_bitConvIntToReal(intValue:IN)

Return value:

Real number converted from an integer.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer to convert to a real number.

6.8.2.9 *val_bitConvBufferCopy(...)*

Description:

Copy a byte array.

Function header:

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

Return value:

Byte array containing the content of the source.

Parameter:

Parameter	Data type	Description
destBuffer[]	CHAR	Byte array as destination.
sourceBuffer[]	CHAR	Byte array as source.

6.8.3 Byte-swapping

6.8.3.1 val_byteSwap(...)

Description:

Permutation of a byte array of size 4 according to the little- / big-endian convention.

IN[]	OUT[]
1	4
2	3
3	2
4	1

Function header:

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

Return value:

Byte array with swapped elements.

Parameter:

Parameter	Data type	Description
inBytes[]	CHAR	Byte array whose elements are swapped.

Example:

```
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(...)**

Description:

Swapping the bytes of an integer according to the little- / big-endian convention.

Function header:

GLOBAL DEFFCT INT val_byteSwapInteger(intValue:IN)

Return value:

Integer converted to little- / big-endian.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer to convert.

Example:

```

INT intValue, iResult

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

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

```

6.8.4 Data validation

6.8.4.1 **val_signalOverflow(...)**

Description:

Checking whether the value assignment on a signal agreement exceeds the scope of the signal agreement.

Function header:

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

Return value:

- TRUE: Value assignment exceeds range of signal agreement.
- FALSE: Value assignment within the range limits of the signal.

Parameter:

Parameter	Data type	Description
intValue	INT	Integer to convert.
bitSize	INT	Width of signal.
numSignum	ENUM	Representation of the signal: • #signed: Signed representation as two's complement. • #unsigned: Represented as an unsigned integer.

Example:

```
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(...)*

Description:

Validation of a number on interval boundaries. In the event of a range violation, a notification message is issued.

Function header:

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

Return value:

- TRUE: Number is in the range.
- FALSE: Number is not in range.

Parameter:

Parameter	Data type	Description
paramName[]	Char	Parameter name (<=24 Zeichen).
paramValue	Real	Parameter.
lowerBoundary	Real	Lower limit of the interval.
upperBoundary	Real	Upper limit of the interval.

location	Enum	Definition of whether the number should be checked inside or outside the interval: • #inside: Checking whether the number is within the interval. • #outside: Checking whether the number is outside the interval.
----------	------	---

Example:

```

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(...)**

Description:

Validating a number against a comparison value. In the event of a violation, a warning message is issued.

Function header:

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

Return value:

- TRUE: The condition for the comparison is met.
- FALSE: The condition for the comparison is not met.

Parameter:

Parameter	Data type	Description
paramName[]	Char	Parameter name (<=24 Zeichen).
paramValue	Real	Parameter.
relOperator	Enum	Comparison operator: • #EQUAL, check for equality. • #UNEQUAL, check for inequality. • #GREATER, check for greater. • #LESS, check for smaller. • #GREATEREQUAL, Check for greater than or equal to. • #LESSEQUAL, Check for less than or equal to.
compValue	Real	Comparison value

Example:

```

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-Events

Providing event routines of the submit and robot interpreter.

7.1 KRL-Modules

The event routines are provided in the modules:

- \R1\TP\g2Framework\g2Interpreter\g2_subAppl.src (submit interpreter events)
- \R1\TP\g2Framework\g2Interpreter\g2_robAppl.src (robot interpreter events)

7.2 Event Summary

Ereignis	Interpreter	Description
sub_OnInitCold()	Submit	Invoke after selecting the submit interpreter.
sub_OnInitHot()	Submit	Invoke after the start of the submit interpreter was previously stopped manually.
sub_OnLoop()	Submit	Cyclic invoke.
sub_OnTimerEvent()	Submit	Cyclic invoke after a period of time in seconds has elapsed. The time is configured via the sub_i_eventTime variable.
sub_OnBlockSel()	Submit	Block selection in a robot program.
sub_OnRobProgCancel()	Submit	Invoke after deselection of a robot program.
sub_OnRobProgReset()	Submit	Invoke after resetting a robot program.
sub_OnEdgeRising()	Submit	Invoked on a rising edge of the sub_b_edgeEvent variable.
sub_OnEdgeFalling()	Submit	Invoked on a falling edge of the sub_b_edgeEvent variable.
sub_OnConfigChanged()	Submit	Invoked when a configuration changes in the ApplicationView.
rob_OnProgInit()	Roboter	Program initialization. Processing is done by invoking rob_ProgInit()
rob_OnProgStart()	Roboter	Invocation when starting a robot program (\$PRO_STATE1 == #P_ACTIVE), provided the robot is on the programmed path. Requirement: rob_ProgInit() has been run beforehand.
rob_OnCycFlagEdgeRising()	Roboter	Invoked with a positive edge change of \$CYCFLAG[161]. Requirement: rob_ProgInit() has been run beforehand.
rob_OnCycFlagEdgeFalling()	Roboter	Invoked with a negative edge change of \$CYCFLAG[161]. Requirement: rob_ProgInit() has been run beforehand.

7.3 Submit-Diagnosis

The framework provides the following diagnostic variables for the submit interpreter in the ApplicationView.

Menue: Display → g2Framework → ApplicationView

Selection: Application → g2Framework; Component → g2_submit

Variable	Description
sub_c_initColdDate[]	Time of the last (cold) start of the submit interpreter.
sub_i_CycleCount	Counter that is incremented with each cycle of the submit.
sub_i_CycleTime	Current cycle time of the submit.
sub_st_cycleTime.min	Minimum cycle time of the submit.
sub_st_cycleTime.max	Maximum cycle time of the submit.

8 Visualization

8.1 ApplicationView

Application overview

Application: Component:

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.	-	

Update
Value
Save
Description

Variables Inputs Outputs

8.1.1 Functionality

The "ApplicationView" plugin offers the following functionalities:

- Display and changes of the states of KRL variables.
- Display of inputs and outputs.
- Changing outputs or signal agreements on outputs.
- Cyclic update.

- Possibility to describe the inputs, outputs and variables.
- Allocation of variables, inputs and outputs to applications and their components via configuration files.
- Definition of limit values for variables.

A sample configuration is part of the installation.

8.1.2 Configuration

The configuration of an application consists of two files:

- Application definition.
- Description file (optional).

8.1.2.1 Application definition

Filename syntax: AppViewConfig_{Applikationsname}.xml

Target directory on the controller: C:\KRC\TP\g2Framework\AppViewConfig\

Note: The file is read when ApplicationView is called.

Application name:

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

Component name:

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

Selection condition for the component:

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

The attributes Trigger, Operator, Value can be used to control whether the components are made available to the user for selection.

Trigger:

Globally declared KRL variable of data type integer.

Operator:

Three operators are allowed:

- „=" if trigger equals the value, the component is offered for selection.
- „<“ if trigger is less than the value, the component is offered for selection.
- „>“ if trigger is greater value, the component is offered for selection.

Value:

Comparison value for the trigger variable.

variable definition

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

Variable name:

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

Area limits:

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

Unit:

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

Selection (Data type Enum):

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

User group:

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

- Config="User": User group User can change variable states.
- Config="Expert": Expert user group can change variable states.
- Config="NoConfig": Changing variable states is not possible.

Definition of inputs and outputs

Input/Output Names:

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

The names of digital inputs and outputs are integer variables that are used as an index for the input or output.

Signal agreements must be declared globally.

Unit:

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

In the case of signal agreements, the unit is not shown in the display.

Area limits:

Area limits are checked for outputs during signal agreements. The maximum value is limited by the signal width. If no max value is configured, the value resulting from the signal width is used.

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

User group:

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

- Config="User": User group User can change IO states.
- Config="Expert": Expert user group can change IO states
- Config="NoConfig": Changes of IO states are not possible

8.1.2.2 Description file

Descriptions of the variables are defined here. The descriptions can be stored in multiple languages. The file is optional.

Filename syntax: AppViewConfig_{Applikationsname}.kxr

Directory on the controller: C:\KRC\TP\g2Framework\Kxr\

The file is read after a cold start of the controller.

Application name:

The name must be identical to the file name (without the ".kxr" extension).

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

Variable or signal names:

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

Description:

```
<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>

```

The description can be extended in all languages supported by the HMI.

8.1.2.3 Variable declaration in KRL

KRL variables and signal declarations must be declared globally.

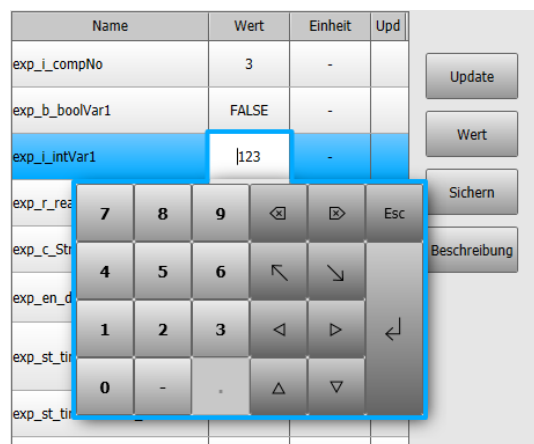
8.1.3 Operation

8.1.3.1 Menu

Main menu: Dispaly → g2Framework → ApplicationView

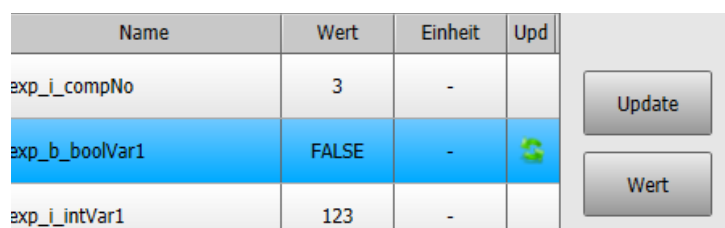
8.1.3.2 Change variable value

- Tab „Variables“.
- Select variable.
- Button „Value“.
- Select the value of the variable.
- Set value.
- "Save" button



8.1.3.3 Update tag value cyclically

- Tab „Variables“
- Select variable
- Button „Update“



The sign symbolizes the cyclic update: 

A maximum of 10 variables can be updated at the same time. Inputs and outputs are always updated cyclically.

8.1.3.4 Variable description

- Tab „Variables“.
- Select variable.
- Button „Description“.

