

IndraWorks 15VRS

Basic Libraries, IndraLogic 2G

Library
R911398633

Edition 01



Title	IndraWorks 15VRS Basic Libraries, IndraLogic 2G
Type of Documentation	Library
Document Typecode	DOK-IL*2G*-BASLIB**V15-LI01-EN-P
Internal File Reference	RS-807b6ea0f5afb4e60a347e8632e3d462-1-en-US-3
Change Record	Edition 01, 2019-07 Refer to tab. 1-1 "Change Record" on page 1
Copyright	© Bosch Rexroth AG 2019 All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Liability	The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.
Editorial Department	Engineering Automation Systems PLC Engineering, KWa (MiNi/PiaSt)

Table of Contents

	Page
1 About this documentation.....	1
1.1 Validity of the documentation.....	1
1.2 Basic libraries, IndraLogic 2G - overview.....	1
1.3 Using safety instructions.....	1
1.3.1 Structure of the safety instructions.....	1
1.3.2 Explaining signal words and safety alert symbol.....	2
1.3.3 Symbols used.....	3
1.3.4 Explaining the signal alert symbol on the device.....	3
1.4 Required and supplementing documentation.....	3
2 Standard interfaces at function blocks.....	5
2.1 Motivation.....	5
2.2 Function block types.....	5
2.3 Block types and their interfaces.....	6
2.4 Inputs and outputs of state-controlled function blocks, final processing.....	7
2.5 Inputs and outputs of edge-controlled function blocks, final processing.....	10
2.6 Inputs and outputs of state-controlled function clocks, permanent processing.....	13
2.7 Inputs and outputs of edge-triggered function blocks, permanent processing.....	16
3 RIL_CommonTypes.library.....	19
3.1 Introduction and overview.....	19
3.2 Global_Variables.....	20
3.2.1 IL_NULL.....	20
3.3 Data types to reference objects of Motion function blocks	20
3.3.1 Referencing objects, introduction and overview.....	20
3.3.2 CONTROLS.....	21
3.3.3 AXIS_REF.....	21
3.3.4 OBJECTS.....	22
3.3.5 AXIS_GROUP_REF.....	23
3.3.6 KIN_OBJECTS.....	24
3.3.7 TOUCHPROBE_REF.....	25
3.3.8 TP_OBJECTS.....	25
3.4 Data types to support cams, MotionProfiles and FlexProfiles	26
3.4.1 Data type to switch between cams, general information.....	26
3.4.2 MC_CAM_ID.....	26
3.4.3 MC_STEP_MODE, MotionProfile extended to FlexProfile	26
3.5 Data types to synchronize in the position-controlled synchronous operation modes	50
3.5.1 Synchronization, introduction and overview.....	50
3.5.2 MC_START_MODE.....	51
3.5.3 MC_SYNC_DIRECTION.....	53
3.6 Data types to diagnose errors in the Motion function blocks	54
3.6.1 ERROR_CODE.....	54
3.6.2 ERROR_STRUCT.....	55

Table of Contents

	Page
3.6.3 ERROR_TABLE.....	56
3.7 Data types to report date and absolute time specifications for Motion function blocks	57
3.7.1 Reporting date and absolute time specification, introduction and overview.....	57
3.7.2 SYS_TIME64.....	57
3.7.3 EXT_SYS_TIME_DATE.....	57
3.7.4 SYS_TIME_DATE.....	58
3.8 Data types to support touch probe functions	58
3.8.1 Data types to support touch probe functions, general information.....	58
3.8.2 MB_DIFF_EVAL.....	59
3.8.3 MB_EXPECT_WINDOW_MODE.....	60
3.8.4 MB_INTERPOLATION_MODE.....	60
3.8.5 MB_POSITION_EVAL.....	61
3.8.6 MB_PROBE_MODULE.....	61
3.8.7 MB_PROBE_INPUT.....	61
3.8.8 MB_PROBE_DATA.....	62
3.8.9 MB_PROBE_CONFIG.....	63
3.8.10 MB_PROBE_CONFIG, special features of Sercos-based axes.....	63
4 RIL_Utilitylibrary.....	65
4.1 Introduction and overview.....	65
4.2 Jogging variables (adjustment).....	69
4.2.1 Introduction and overview.....	69
4.2.2 IL_ContinuousAdjustType01.....	72
4.2.3 IL_ContinuousAdjustType02.....	75
4.2.4 IL_IncrementalAdjustType01.....	78
4.2.5 Parameterizing function blocks to jog variables.....	80
4.3 Basic classes for the function block development (Base FB).....	80
4.3.1 Introduction and overview.....	80
4.3.2 Basic function block IL_EnableDoneBase.....	86
4.3.3 Basic function block IL_EnableInOpBase.....	89
4.3.4 Basic function block IL_EnableInOpShutdownBase.....	91
4.3.5 Basic function block IL_ExecuteDoneBase.....	94
4.3.6 Basic function block IL_ExecuteInOpBase.....	97
4.3.7 Basic function block IL_ExecuteDoneAbortedBase.....	100
4.3.8 Basis function block IL_ExecuteDoneAbortedShutdownBase.....	102
4.3.9 IL_FB_STATES.....	107
4.4 Conversion (bit-stream connect).....	107
4.4.1 Introduction and overview.....	107
4.4.2 IL_BoolToByte.....	108
4.4.3 IL_BoolToDWord.....	109
4.4.4 IL_BoolToWord.....	111
4.4.5 IL_ByteToDWord.....	112
4.4.6 IL_ConcatByte.....	113
4.4.7 IL_ConcatWord.....	114
4.5 Conversion (bit-stream split).....	115
4.5.1 Introduction and overview.....	115

	Page
4.5.2 IL_ByteToBool.....	115
4.5.3 IL_DWordToBool.....	116
4.5.4 IL_DWordToByte.....	119
4.5.5 IL_HighByte.....	119
4.5.6 IL_HighWord.....	120
4.5.7 IL_LowByte.....	120
4.5.8 IL_LowWord.....	121
4.5.9 IL_WordToBool.....	121
4.6 Conversion (ByteSwap).....	123
4.6.1 Introduction and overview.....	123
4.6.2 IL_SwapDWord.....	123
4.6.3 IL_SwapWord.....	124
4.7 Checksums.....	124
4.7.1 Introduction and overview.....	124
4.7.2 IL_CRC16.....	125
4.8 Conversion (data type change).....	126
4.8.1 Introduction and overview.....	126
4.8.2 IL_AsciiToByte.....	126
4.8.3 IL_ByteToAscii.....	126
4.8.4 IL_RealToString.....	127
4.8.5 IL_LRealToString.....	127
4.9 Debugging.....	129
4.9.1 Introduction and overview.....	129
4.9.2 IL_HighResTimeDiff.....	129
4.9.3 IL_HighResTimeTick.....	130
4.10 FlipFlops.....	131
4.10.1 Introduction and overview.....	131
4.10.2 IL_Toggle.....	131
4.11 Mathematical functions.....	133
4.11.1 Introduction and overview.....	133
4.11.2 IL_InterpolationLinear.....	133
4.11.3 IL_MaxValue.....	135
4.11.4 IL_COORD_PAIR_TYPE01.....	136
4.12 Character strings without length limitation (StringFun_NoLimitStrLength).....	136
4.12.1 Introduction and overview.....	136
4.12.2 IL_StrLower.....	139
4.12.3 IL_StrUpper.....	140
4.12.4 IL_StrCompare.....	141
4.12.5 IL_StrCompareNoCase.....	141
4.12.6 IL_StrNCompare.....	142
4.12.7 IL_StrNCompareNoCase.....	143
4.12.8 IL_AsciiStreamToBase64.....	143
4.12.9 IL_Base64ToAsciiStream.....	145
4.12.10 IL_BufferToString.....	147
4.12.11 IL_StringToBuffer.....	148
4.12.12 IL_StrCopy.....	150

Table of Contents

	Page
4.12.13 IL_StrNCopy.....	151
4.12.14 IL_CONCAT.....	152
4.12.15 IL_DELETE.....	153
4.12.16 IL_FIND.....	154
4.12.17 IL_INSERT.....	154
4.12.18 IL_LEFT.....	155
4.12.19 IL_LEN.....	156
4.12.20 IL_MID.....	156
4.12.21 IL_REPLACE.....	157
4.12.22 IL_RIGHT.....	158
4.12.23 IL_Ascii_To_Char.....	159
4.12.24 IL_Char_To_Ascii.....	160
4.12.25 IL_Int_To_Str.....	160
4.12.26 ENUM_INT_OPTION.....	161
4.12.27 IL_Real_To_Str.....	161
4.12.28 ENUM_REAL_OPTION.....	162
4.12.29 IL_String_To_Unicode.....	162
4.12.30 ENUM_OPT_UTF.....	163
4.12.31 IL_Unicode_To_String.....	163
4.12.32 IL_BufferGetElementCount.....	164
4.12.33 IL_Code.....	165
4.12.34 IL_StrInverse.....	166
4.12.35 IL_StrSetEndOfString.....	166
4.12.36 IL_StrGetElement.....	167
4.12.37 IL_StrLChr.....	168
4.12.38 IL_StrRChr.....	169
4.12.39 IL_StrSetElement.....	170
4.12.40 IL_StrSplit.....	171
4.12.41 IL_StrTrimAll.....	172
4.12.42 IL_StrTrimSpace.....	173
4.12.43 ENUM_TRIM_OPT.....	174
4.13 Time and date calculation, version 2 (time).....	174
4.13.1 Introduction and overview.....	174
4.13.2 IL_Date.....	175
4.13.3 IL_DateAndTime.....	175
4.13.4 IL_ExtSysTimeDate.....	176
4.13.5 IL_LocalTime.....	176
4.13.6 IL_SysTime64.....	178
4.13.7 IL_SysTime64ToSysTimeDate.....	179
4.13.8 IL_SysTimeDate.....	179
4.13.9 IL_SysTimeDateToSysTime64.....	180
4.13.10 IL_TimeOfDay.....	180
4.13.11 IL_TIMEZONE.....	181
4.14 Double-linked lists (Utilities/Logic).....	184
4.14.1 Introduction and overview.....	184
4.14.2 IL_LinkedList.....	184

	Page
4.14.3 IL_LINKED_LIST_MAINTENANCE.....	188
4.14.4 IL_LINKED_LIST_MODE.....	189
4.15 Modulo value calculation (Utilities/Logic).....	189
4.15.1 Introduction and overview.....	189
4.15.2 IL_GetModuloType01.....	190
4.15.3 IL_GetModuloType02.....	191
4.15.4 IL_GetModuloType03.....	192
4.15.5 IL_GetModuloType04.....	193
5 RIL_ModbusTCP.library.....	195
5.1 Introduction and overview.....	195
5.2 ModbusTCP server.....	196
5.2.1 ModbusTCP server applications.....	196
General information.....	196
Configuring with Siemens WinCC-flexible 2007.....	198
Configuring with Pro-face GP-Pro EX 2.2.....	202
5.2.2 IL_ModbusTCPServer.....	209
5.2.3 IL_ModbusTCPServerType02.....	212
5.2.4 IL_MODBUS_STATS.....	218
5.3 ModbusTCP client.....	218
5.3.1 Introduction and overview.....	218
5.3.2 IL_ModbusTCPClient.....	219
5.3.3 IL_ModbusTCPReadCoils.....	221
5.3.4 IL_ModbusTCPReadDiscreteInputs.....	224
5.3.5 IL_ModbusTCPReadHoldingRegisters.....	226
5.3.6 IL_ModbusTCPReadInputRegisters.....	228
5.3.7 IL_ModBusTCPWriteSingleCoil.....	230
5.3.8 IL_ModBusTCPWriteSingleRegister.....	232
5.3.9 IL_ModBusTCPWriteMultipleCoils.....	234
5.3.10 IL_ModBusTCPWriteMultipleRegisters.....	236
5.3.11 IL_ModBusTCPReadWriteMultipleRegisters.....	238
5.3.12 IL_ModbusTCPReadDeviceIdentification.....	241
6 RIL_SocketComm.library.....	245
6.1 Introduction and overview.....	245
6.2 I/O time response of the function blocks.....	245
6.3 Functional description.....	246
6.3.1 TCP.....	246
6.3.2 UDP.....	250
6.3.3 ICMP.....	256
6.3.4 FTP.....	256
6.3.5 TFTP.....	257
6.3.6 Cyclic communication.....	257
6.3.7 Byte order in IP-based communication.....	257
6.3.8 Security - Disable unsecure protocols.....	258

Table of Contents

	Page
6.4 TCP asynchronous function blocks.....	258
6.4.1 Introduction and overview.....	258
6.4.2 IL_TCPConnectionAsync.....	259
6.4.3 IL_TCPInitialAsync.....	263
6.4.4 IL_TCPSendAsync.....	267
6.4.5 IL_TCPRecvAsync.....	268
6.4.6 IL_TCPRecvPacketAsync.....	270
6.4.7 IL_TCPCloseAsync.....	272
6.5 TCP synchronous functions.....	274
6.5.1 Introduction and overview.....	274
6.5.2 IL_TCPInitial.....	274
6.5.3 IL_TCPSend.....	276
6.5.4 IL_TCPRecv.....	277
6.5.5 IL_TCPRecvPacket.....	278
6.5.6 IL_TCPClose.....	279
6.6 UDP asynchronous function blocks.....	279
6.6.1 Introduction and overview.....	279
6.6.2 IL_UDPSocketAsync.....	280
6.6.3 IL_UDPInitialAsync.....	282
6.6.4 IL_UDPSendtoAsync.....	284
6.6.5 IL_UDPRecvfromAsync.....	286
6.6.6 IL_UDPCloseAsync.....	287
6.7 UDP synchronous functions.....	289
6.7.1 Introduction and overview.....	289
6.7.2 IL_UDPInitial.....	289
6.7.3 IL_UDPSendto.....	290
6.7.4 IL_UDPRecvfrom.....	291
6.7.5 IL_UDPClose.....	292
6.8 Asynchronous ICMP function blocks.....	293
6.8.1 IL_PingAsync.....	293
6.9 Synchronous ICMP functions.....	295
6.9.1 IL_Ping.....	295
6.10 Asynchronous FTP function blocks.....	295
6.10.1 Introduction and overview.....	295
6.10.2 IL_FTPGetAsync.....	296
6.10.3 IL_FTPPutAsync.....	298
6.10.4 IL_FTPGetFileAsync.....	301
6.10.5 IL_FTPPutFileAsync.....	304
6.11 Asynchronous TFTP function blocks.....	306
6.11.1 Introduction and overview.....	306
6.11.2 IL_TFTPGetFileAsync.....	306
6.11.3 IL_TFTPPutFileAsync.....	309
6.12 Asynchronous communication function blocks for cyclic communication.....	311
6.12.1 Introduction and overview.....	311
6.12.2 IL_TCPCyclic.....	311
6.12.3 IL_UDPCyclic.....	316

	Page
6.12.4 IL_CONN_MODE.....	320
6.12.5 IL_CONN_STATUS.....	321
6.12.6 ETHERNET_TABLE, ERROR_TABLE 16#0150	322
ETHERNET_TABLE, General Information.....	322
ETHERNET_TABLE, General, 16#00000001-16#00000212.....	322
ETHERNET_TABLE, IL_TcpCyclic, IL_UdpCyclic, 16#00001001-16#00001002.....	324
ETHERNET_TABLE, IL_FTPGetAsync, IL_FTPPutAsync, IL_FTPGetFileAsync, IL_FTP_PutFileAsync, 16#00002001-16#00002004.....	324
ETHERNET_TABLE, IL_TFTPGetFileAsync, IL_TFTPPutFileAsync, 16#00002101-16#00002108.....	325
7 RIL_LoopControl.library.....	327
7.1 Introduction and overview.....	327
7.2 Time-discrete function blocks.....	329
7.3 Event-controlled function blocks.....	330
7.4 Controllers.....	330
7.4.1 Introduction and overview.....	330
7.4.2 IL_PIDType01.....	331
7.4.3 IL_PIDType02.....	334
7.4.4 IL_PIDType03.....	338
7.4.5 IL_PIDType04.....	342
7.4.6 IL_PIType01.....	346
7.4.7 IL_PIType02.....	350
7.5 Elements (basic elements).....	354
7.5.1 Introduction and overview.....	354
7.5.2 IL_DeadtimeType01.....	355
7.5.3 IL_DeadtimeType02.....	357
7.5.4 IL_DeviationResponseCurve.....	360
7.5.5 IL_DT1Type01.....	362
7.5.6 IL_DT1Type02.....	364
7.5.7 IL_PDT1Type01.....	367
7.5.8 IL_PDT1Type02.....	370
7.5.9 IL_PT1Type01.....	373
7.5.10 IL_PT1Type02.....	376
7.5.11 IL_TransportDelayType01.....	378
7.5.12 IL_TransportDelayType02.....	381
7.5.13 IL_ValueLimit.....	384
7.5.14 IL_ValueWindow.....	385
7.6 Filter.....	387
7.6.1 Introduction and overview.....	387
7.6.2 IL_FIRType01.....	387
7.6.3 IL_FIRType02.....	389
7.6.4 IL_IIRType01.....	391
7.6.5 IL_IIRType02.....	394
7.6.6 IL_StateVariableType01.....	396
7.6.7 IL_StateVariableType02.....	398

Table of Contents

	Page
7.6.8 IL_FILTER_TYPE.....	400
7.6.9 IL_FIR_COEFF.....	401
7.6.10 IL_IIR_COEFF.....	401
7.7 Generators (signal generators).....	401
7.7.1 Introduction and overview.....	401
7.7.2 IL_NoiseGeneratorType01.....	402
7.7.3 IL_NoiseGeneratorType02.....	404
7.7.4 IL_PRBSGeneratorType01.....	405
7.7.5 IL_PRBSGeneratorType02.....	407
7.7.6 IL_AngleGeneratorType01.....	410
7.7.7 IL_AngleGeneratorType02.....	411
7.7.8 IL_PhaseController.....	412
7.7.9 IL_SignalGeneratorType01.....	414
7.7.10 IL_SignalGeneratorType02.....	418
7.7.11 IL_AmplitudeControllerType01.....	422
7.7.12 IL_AmplitudeControllerType02.....	425
7.7.13 IL_AMP_CONTROL.....	428
7.7.14 IL_CENTERLINE_CONTROL.....	428
7.7.15 IL_CURVETYPE.....	429
7.8 Statistics (statistic function blocks).....	429
7.8.1 Introduction and overview.....	429
7.8.2 IL_HistogramType01.....	429
7.8.3 IL_HistogramType02.....	431
7.8.4 IL_StatisticType01.....	433
7.8.5 IL_StatisticType02.....	435
7.8.6 IL_SuppressionType01.....	438
7.8.7 IL_SuppressionType02.....	442
7.8.8 IL_SuppressionType03.....	447
7.8.9 IL_SuppressionType04.....	452
7.9 Temperature controller (TemperatureController).....	458
7.9.1 Introduction and overview.....	458
7.9.2 IL_TempControlType01.....	458
7.9.3 IL_TempControlType02.....	476
7.9.4 IL_TEMPCONTROL_CONTR_PARAM.....	486
7.9.5 IL_TEMPCONTROL_CONTROL_MODE.....	486
7.9.6 IL_TEMPCONTROL_CONTROL_TYPE.....	487
7.9.7 IL_TEMPCONTROL_EXTENDED_SETTINGS.....	487
7.9.8 IL_TEMPCONTROL_LIMITS.....	488
7.9.9 IL_TEMPCONTROL_MODEL_PARAM.....	489
7.10 Two-position controller.....	489
7.10.1 IL_TwoPosCtrlType01.....	489
7.11 Utilities.....	491
7.11.1 IL_COORD_TYPE01.....	491
7.11.2 IL_SUPPRESSION_LIMITS.....	491
7.11.3 IL_SUPPRESSION_MODE.....	492

	Page
8 RIL_HMI_Utility.library.....	493
8.1 Introduction and overview.....	493
8.2 IL_MKeys.....	493
8.3 IL_UdpKeys.....	494
9 IL_VCP_DP.library.....	497
9.1 Introduction and overview.....	497
9.2 VCP_PBS16_A4096	497
9.3 VCP_PBS32_A4096	498
9.4 VCP_PBS32_A65536	499
10 Service and support.....	501
Index.....	503

1 About this documentation

Editions of this documentation

Edition	Release Date	Note
01	2019-07	First edition for version 15VRS

Tab. 1-1: Change Record

1.1 Validity of the documentation

Target group This documentation is intended to inform on the basic libraries of the MLC and MTX systems.

Required qualification: Individual who is able to assess the tasks assigned and to identify possible safety risks owing to qualification in the subject, knowledge and experience. The individual should also be familiar with the standards and regulations.

1.2 Basic libraries, IndraLogic 2G - overview

The **IndraLogic 2G basic libraries** are described in the following, sorted by systems **MLC**, **MTX** und **MLD**:

Name	MTX	MLC	MLD
RIL_CommonTypes.library, page 19 ,	x	x	
RIL_Uilities.library, page 65 , general IL functions and IL function blocks		x	
RIL_ModbusTCP.library, page 195 , diagnostic and communication services for ModbusTCP	x	x	x
RIL_SocketComm.library, page 245 , functions to communicate via TCP/IP and UDP	x	x	
RIL_LoopControl.library, page 327 , controller function blocks	x	x	x
RIL_NetXLoad, for internal use only		x	
RIL_HMI_Uilities.library, page 493 , functions to load HMI M-keys	x	x	
IL_VCP_DP.library, page 497 , functions to trigger small control panels (VCP) via Profibus-DP			

MLC Library requires basic function package PLC

Tab. 1-2: Basic libraries and systems supported by them

1.3 Using safety instructions

1.3.1 Structure of the safety instructions

The safety instructions are structured as follows:

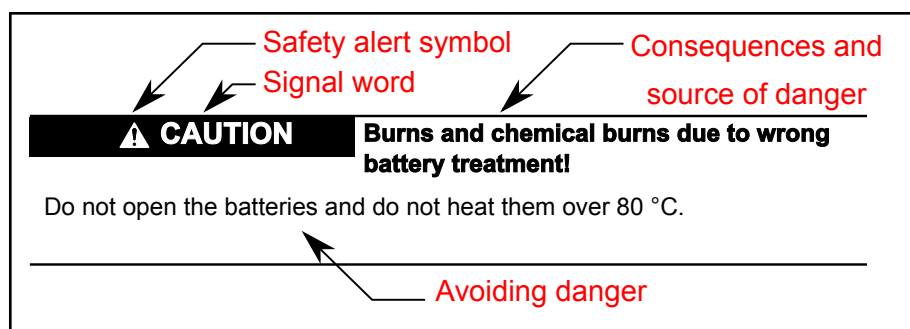


Fig. 1-1: Structure of the safety instructions

1.3.2 Explaining signal words and safety alert symbol

The safety instructions in this documentation contain specific signal words (danger, warning, caution, notice) and, if necessary, a safety alert symbol (according to ANSI Z535.6-2006).

The signal word draws attention to the safety instruction and indicates the risk potential.

The safety alert symbol (triangular safety reflector with exclamation marks), preceding the signal words Danger, Warning, Caution indicates hazards for persons.

DANGER

In case of non-compliance with this safety instruction, death or serious injury **will** occur.

WARNING

In case of non-compliance with this safety instruction, death or serious injury **can** occur.

CAUTION

In case of non-compliance with this safety instruction, minor or moderate injury can occur.

NOTICE

In case of non-compliance with this safety instruction, material damage can occur.

1.3.3 Symbols used

Pointers are displayed as follows:



This is a note.

Tips are displayed as follows:



This is a tip.

1.3.4 Explaining the signal alert symbol on the device



If this symbol is on your device, you have to observe the documentation on the device. The respective documentation informs on the type of hazard as well as the steps required to avoid this hazard.

1.4 Required and supplementing documentation

PLC	MTX	MLC
IndraWorks 15VRS IndraLogic 2G PLC Programming System DOK-IWORKS-IL2GPRO*V15-APRS-EN-P, R911396137 This documentation describes the PLC programming tool IndraLogic 2G and its use. The documentation includes the basic use, first steps, visualization, menu items and editors.	x	x
Field buses	MTX	MLC
MLC 15VRS PLCopen Libraries DOK-XLCMLC-FUNLIB**V15-LIRS-EN-P, R911393307 This documentation describes the function blocks, functions and data types of the RIL_Common-Types, ML_Base and ML_PLCopen libraries for the MLC system. The documentation also includes the error reactions of the function blocks.		x
IndraWorks 15VRS Field Buses DOK-IWORKS-FB*****V15-APRS-EN-P, R911393284 This documentation describes the field bus and local periphery connections supported by the MLC and MTX systems. The focus of this documentation lies in the configuration, parameterization, commissioning and diagnostics of different periphery connections. It is the basis for the online help.	x	x
IndraWorks 15VRS - Field Bus Libraries DOK-IWORKS-FB*LIB**V15-LIRS-EN-P, R911393275 This manual describes the field bus libraries: RIL_ProfibusDP_02, RIL_ProfibusDP Slave, RIL_ProfinetIO, RIL_ProfinetIODevice, RIL_EtherNetIPAdapter, RIL_MappingList, RIL_SERCOSIII, RIL_Inline including their diagnostics and error reactions of the function blocks.	x	x

2 Standard interfaces at function blocks

2.1 Motivation

The majority of the function blocks disposes of an input for the activation and an output for displaying the correct processing.

In addition, an output is often necessary which indicates the processing time.

Additional outputs are defined for the indication of errors.

A uniform naming convention as well as an identical standard interface behavior, increases understanding, shortens the time needed for familiarization and reduces the workload of the support.

2.2 Function block types

Using function blocks, it is possible to encapsulate complex tasks so they can be used again and to address them via defined interfaces. For this, the processing can be either state-controlled or edge-controlled. Furthermore, a difference is made between tasks which can be processed final and tasks which require permanent intervention once they are started.

State-controlled	If a function block always repeats its task after switch-on after it has reached a defined state, this function block processes "state-controlled".
Edge-controlled	If a function block fulfills its task exactly once after being switched on, this function block processes "edge-controlled".
Final processing	If a block can be finally process its order and afterwards stop to work, this is called "final processing".
Permanent processing	If a block can never conclude its order definitively but continuous intervention is necessary, this is defined as "permanent processing".

In combination, there are four function block types possible:

Control	Processing	Example
State-controlled (Enable)	Final (Done)	Continuous reading of a parameter If reading is completed, the function block has completed its task. The function block repeats this task as long as the control input (Enable) remains set, e.g. MB_ReadParameter
State-controlled (Enable)	Permanent (In....)	Terminable control If the control is active, it processes until the control input (Enable) is reset and thus disabled the control
Edge-controlled (Execute)	Final (Done)	One-time writing of a parameter If writing is completed, the function block has completed its task. A new edge is required at the control input (Execute) to start writing again, e.g. MB_WriteParameter
Edge-controlled (Execute)	Permanent (In....)	Non-terminable control Once the control is started via the control input (Execute), it cannot be disabled via the interfaces of this function block, e.g. MC_MoveVelocity

Tab. 2-1: Overview on function block types

2.3 Block types and their interfaces

Inputs In order to mark whether a function block processes state-controlled or edge-controlled, two different variable names are used for the inputs for the activation of the function block.

- Enable = state-controlled
- Execute = edge-controlled

Outputs To mark whether a function block processes finally or permanently, different variable names are used for the outputs indicating the processing state.



- Done = Final processing
- In..... = Permanent processing (e.g., InOperation, InSync, InVelocity, InGear, InTorque...)

The optional "Active" output can be used with final and permanent processing function blocks. However, the behavior differs!

All outputs for the error identification (Error, ErrorID, ErrorIdent) can be found with final and permanent processing function blocks. Both show the same behavior.

Standardized identifiers

Identifier	Description
Inputs	
Enable	Enable input of state-controlled function blocks
Execute	Enable input of edge-controlled function blocks. Other function blocks can interrupt this function block
ExecuteLock	Enable input of edge-controlled function blocks. As long as this input is TRUE, the function block cannot be interrupted by others
Outputs	
Active	Output marking the processing time
Done	Processing completed successfully and the data outputs are valid
In.....	The output signals that the function block is processing its task and that the data outputs are valid
Shutdown (optional)	The output signals that the function block is currently reaching a defined final state (e.g., it stops axes, releases resources...). The function block has to be called until "Shutdown" becomes FALSE
CommandAborted	Output indicates that the function block was interrupted (e.g. by another function block)
Error	Process completed with error
ErrorID	Output for rough error classification
ErrorIdent	Output for detailed error classification

Tab. 2-2: Overview on standardized identifiers

2.4 Inputs and outputs of state-controlled function blocks, final processing

Control/ processing	I/O	Variable name	Description
State-controlled/final	I	Enable	The input variables are registered with a positive edge of "Enable". New input values are only applied at another positive edge at "Enable". "Enable" has to be TRUE during the complete function block processing! If "Enable" is disabled, processing is interrupted and the "Done", "Active", "CommandAborted" and "Error" outputs are set to FALSE. If required, "Shutdown" signals that further signals are necessary to make the function block to reach its final state defined
	O	Done	If "Done" is TRUE, the function block has successfully completed its order and is then in a final state. Data outputs are valid now. "Active", "Error" and "CommandAborted" are FALSE! "Done" remains TRUE for exactly one PLC cycle. If, afterwards, "Enable" is still TRUE, the function block starts its processing again
	O	Active	If "Active" is TRUE, the function block is processing its actual task and is then in an intermediate state. Possible preprocessing are not marked with this output! "Done", "Error" and "CommandAborted" are FALSE!
	O	Shutdown (optional)	If a "Shutdown" output is available, the function block requires multiple cycles of computation time to disable "Enable" to reach a defined final state and to release the resources used. The function block has to be called until "Shutdown" becomes FALSE!
	O	CommandAborted	If "CommandAborted" is TRUE, the function block has been interrupted and is now in a final state. "Done", "Active" and "Error" are FALSE! "CommandAborted" remains TRUE until the control input "Enable" is disabled. An additional reset input is not necessary. If a "Shutdown" output is available, it becomes simultaneously TRUE with "CommandAborted". In this case, the function block has to be called until "Shutdown" becomes FALSE
	O	Error	If "Error" is TRUE, the function block was interrupted due to an error and is now in a final state. "Error" remains TRUE until the control input "Enable" is disabled. An additional reset input is not necessary. If a "Shutdown" output is available, it becomes simultaneously TRUE with "Error". In this case, the function block has to be called until "Shutdown" becomes FALSE

Tab. 2-3: I/O of state-controlled function blocks, final processing



If it is required that certain inputs are not only applied at the edge at "Enable", but cyclically during the function block processing, it is to be documented explicitly and not to be specified via the variable name!



For function blocks with "Shutdown", define the behavior of the function block when "Enable" is set to TRUE while "Shutdown" remains set (retrigger).

Standard interfaces at function blocks

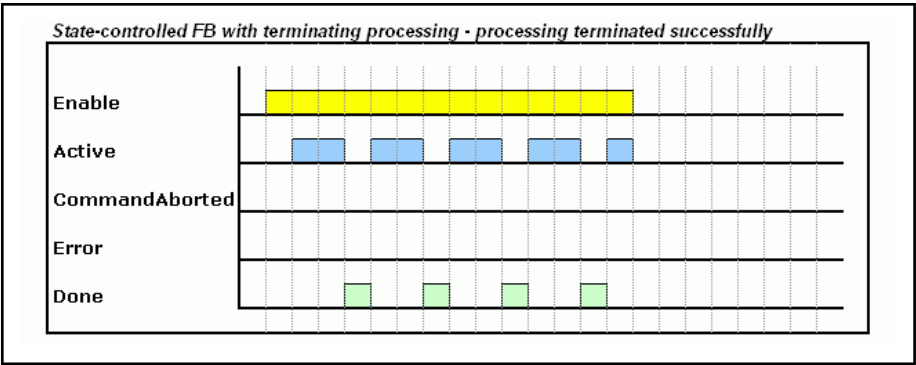


Fig. 2-1: I/O of state-controlled function blocks, final processing without "Shut-down". Processing successfully completed

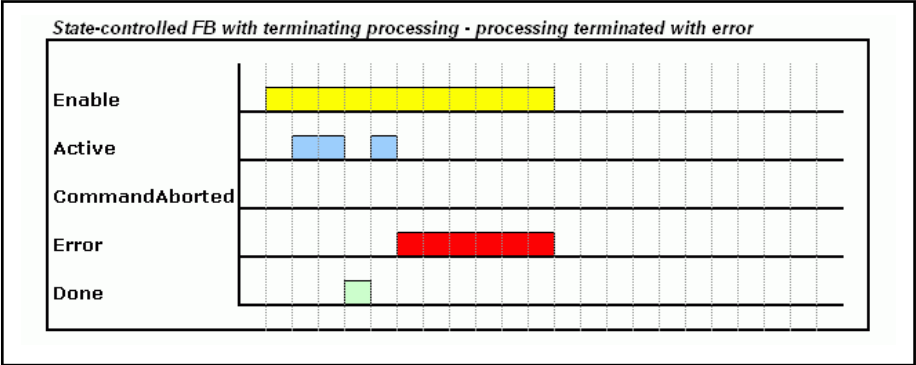


Fig. 2-2: I/O of state-controlled function blocks, final processing without "Shut-down". Processing completed with error

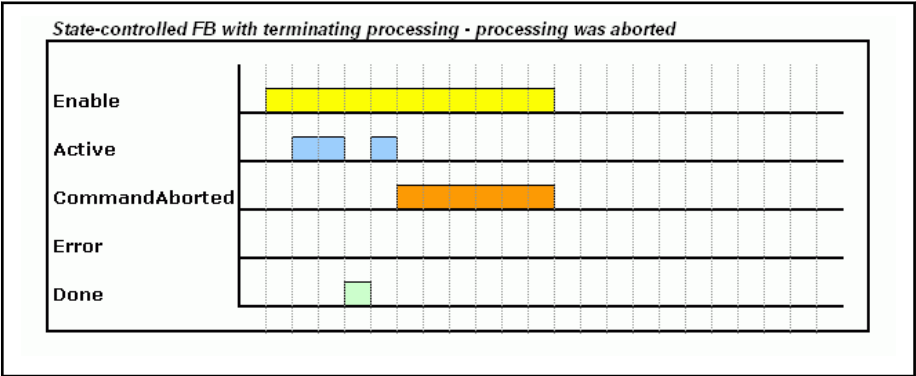


Fig. 2-3: I/O of state-controlled function blocks, final processing without "Shut-down". Processing interrupted

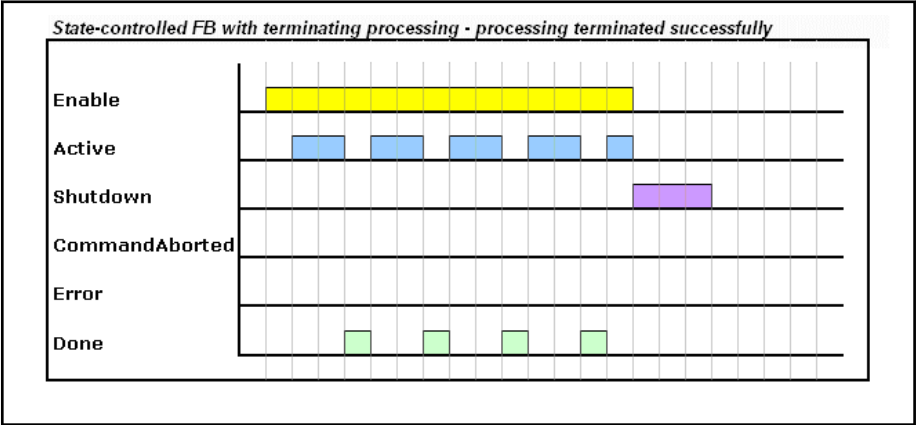


Fig. 2-4: I/O of state-controlled function blocks, final processing with "Shut-down". Processing successfully completed

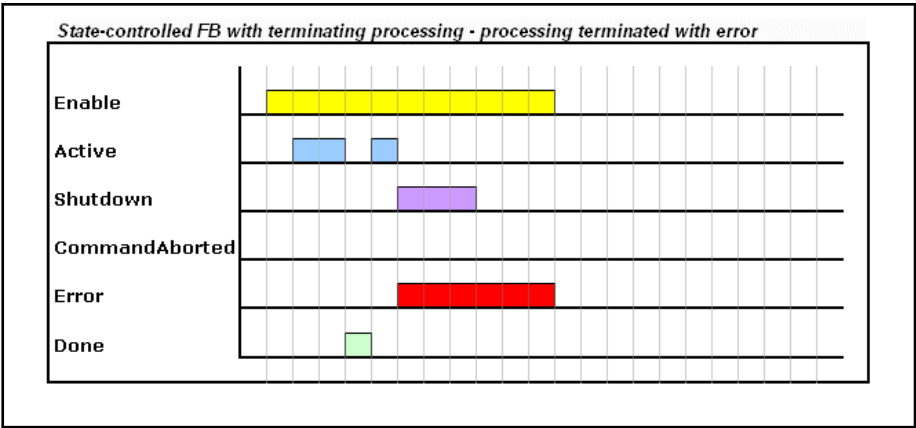


Fig. 2-5: I/O of state-controlled function blocks, final processing with "Shutdown". Processing completed with error

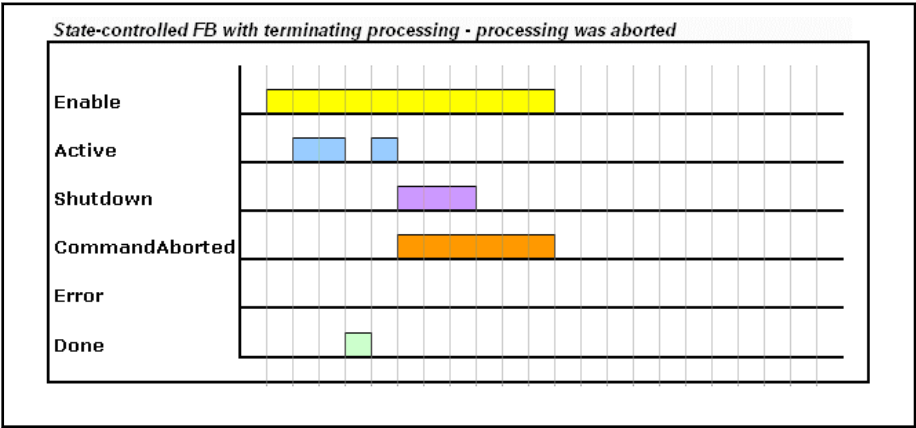


Fig. 2-6: I/O of state-controlled function blocks, final processing with "Shutdown". Processing was interrupted

2.5 Inputs and outputs of edge-controlled function blocks, final processing

Control/ processing	I/O	Name	Description
Edge-con- trolled/final	I	Execute	The input variables are registered with a positive edge of "Execute". New input values are only applied at another positive edge at "Execute". The change of edges at "Execute" suffices to start the block. For further processing, the state of "Execute" is not important. With a new edge change during the processing, the previous task is discarded, the inputs are applied again and the task is continued with the new values (retriggering, subsequent triggering). If a "Shutdown" output is available, retriggering is perhaps not supported. The function block documentation provides clarity on this issue
	O	Done	If "Done" is TRUE, the function block has successfully completed its order and is then in a final state. Data outputs are valid now. "Active", "Error" and "CommandAborted" are FALSE! If the control input "Execute" is FALSE when the order is completed, "Done" remains TRUE for exactly one PLC cycle. If "Execute" is TRUE, "Done" remains TRUE until "Execute" is disabled. If a "Shutdown" output is available, it becomes simultaneously TRUE with "Done". In this case, the function block has to be called until "Shutdown" becomes FALSE
	O	Active	If "Active" is TRUE, the function block is processing its actual task and is then in an intermediate state. Possible preprocessing are not marked with this output! "Done", "Error", "Shutdown" and "CommandAborted" are FALSE!
	O	Shutdown (optional)	If a "Shutdown" output is available, the function block requires multiple cycles of computation time after the completion of the task (signaled via "Done") to reach a defined final state and to release the resources used. The function block has to be called until "Shutdown" becomes FALSE!
	O	CommandAborted	If "CommandAborted" is TRUE, the function block has been interrupted and is now in a final state. "Done", "Active" and "Error" are FALSE! If the control input "Execute" is FALSE at the time when the order is interrupted, "CommandAborted" remains TRUE for exactly one PLC cycle. If "Execute" is TRUE, "CommandAborted" remains TRUE until "Execute" is disabled. If a "Shutdown" output is available, it becomes simultaneously TRUE with "CommandAborted". In this case, the function block has to be called until "Shutdown" becomes FALSE
	O	Error	If "Error" is TRUE, the function block was interrupted due to an error and is now in a final state. If the control input "Execute" is FALSE when the error occurs, "Error" remains TRUE for exactly one PLC cycle. If "Execute" is TRUE, "Error" remains TRUE until "Execute" is disabled. If a "Shutdown" output is available, it becomes simultaneously TRUE with "Error". In this case, the function block has to be called until "Shutdown" becomes FALSE

Tab. 2-4: I/O of state-controlled function blocks, final processing



If certain inputs are not only applied with the 0/1 edge at "Execute", but cyclically during the function block processing, it is to be documented explicitly and not to be specified via the variable name!



For function blocks with "Shutdown", define the behavior of the function block when a new edge is detected at "Execute" while "Shutdown" remains set (retrigger).

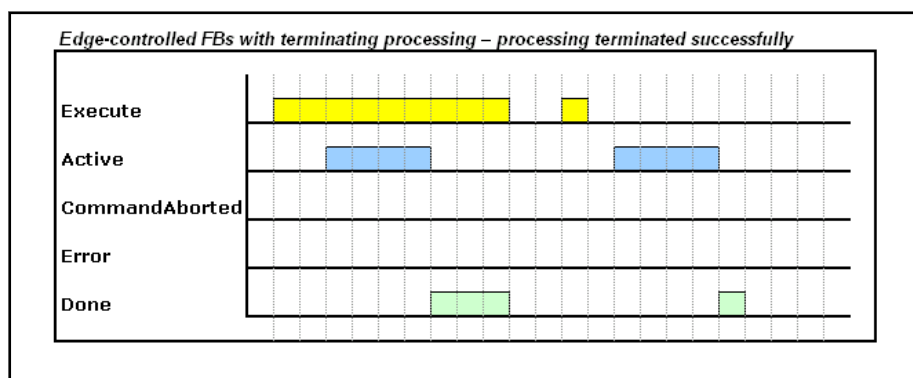


Fig. 2-7: I/O of edge-controlled function blocks, final processing without "Shutdown". Processing successfully completed

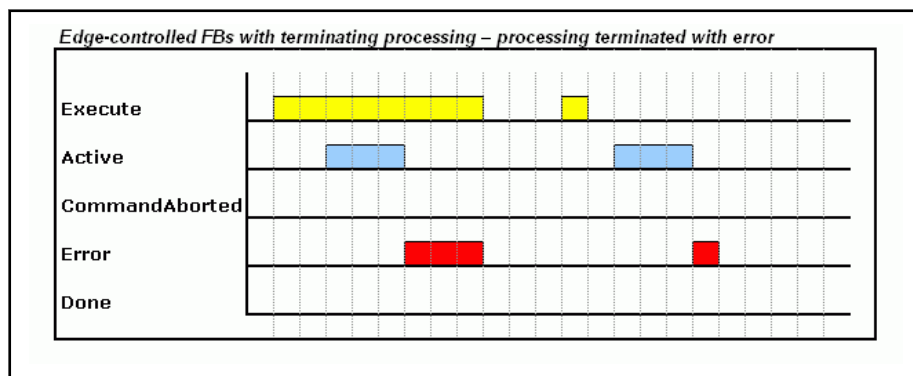


Fig. 2-8: I/O of edge-controlled function blocks, final processing without "Shutdown". Processing completed with error

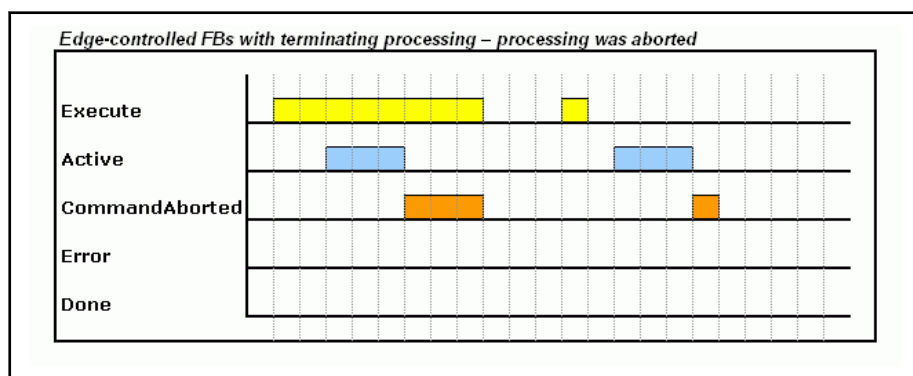


Fig. 2-9: I/O of edge-controlled function blocks, final processing without "Shutdown". Processing interrupted

Standard interfaces at function blocks

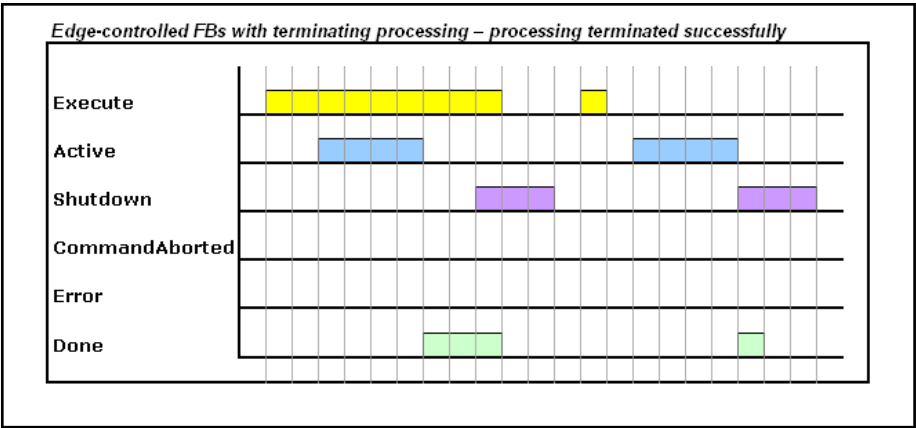


Fig. 2-10: I/O of edge-controlled function blocks, final processing with "Shutdown". Processing successfully completed

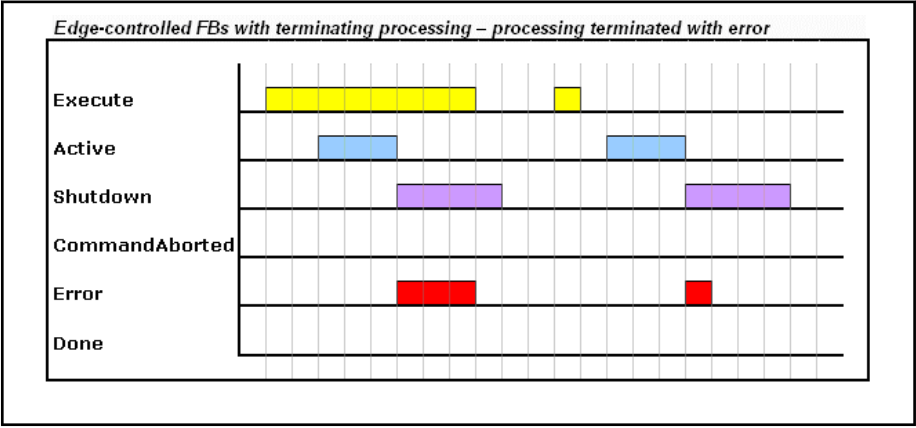


Fig. 2-11: I/O of edge-controlled function blocks, final processing with "Shutdown". Processing completed with error

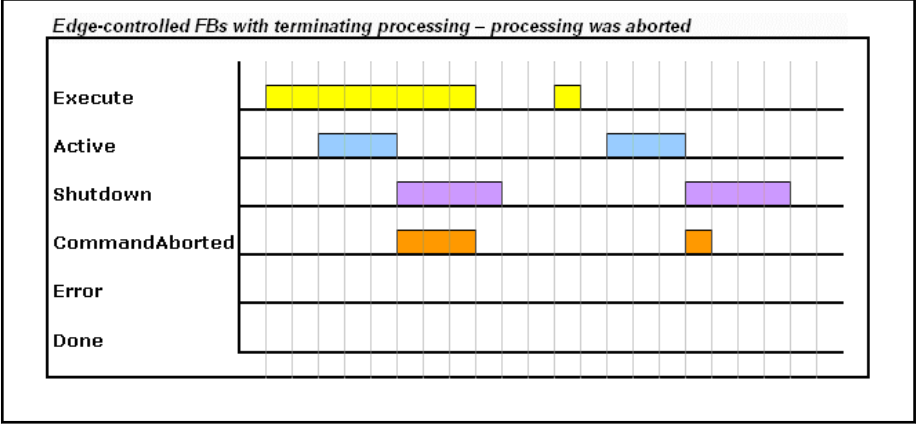


Fig. 2-12: I/O of edge-controlled function blocks, final processing with "Shutdown". Processing interrupted

2.6 Inputs and outputs of state-controlled function clocks, permanent processing

Control/ processing	I/O	Variable name	Description
State-controlled/ permanent	I	Enable	The input variables are registered with a positive edge of "Enable". New input values are only applied at another positive edge at "Enable". "Enable" has to be TRUE during the complete function block processing! If "Enable" is disabled, processing is interrupted and the "In....", "Active", "CommandAborted" and "Error" outputs are set to FALSE. If required, "Shutdown" signals that further signals are necessary to make the function block to reach its final state defined
	O	In....	If "In...." is TRUE, the function block reached its target, but it is still in intervention to "keep" the achieved. Thus, it is thus in a permanent final state. Data outputs are valid now. Since the function block remains in intervention, "Active" also has to remain TRUE. "Error" and "CommandAborted" are FALSE! As long as "Enable" is TRUE, "In...." also remains TRUE
	O	Active	If Active is TRUE, the function block processes its actual task. Possible preprocessing are not marked with this output! Since the function block remains in intervention permanently, "Active" remains TRUE until the function block is switched off via the control input "Enable" or until it is completed by "Error" or "CommandAborted". As long as "Active" is TRUE, "Error" or "CommandAborted" have to be FALSE!
	O	Shutdown (optional)	If a "Shutdown" output is available, the function block requires multiple cycles of computation time to disable "Enable" to reach a defined final state and to release the resources used. The function block has to be called until "Shutdown" becomes FALSE!
	O	CommandAborted	If "CommandAborted" is TRUE, the function block has been interrupted and is now in a final state. "In....", "Active" and "Error" are FALSE! "CommandAborted" remains TRUE until the control input "Enable" is disabled. An additional reset input is not necessary. If a "Shutdown" output is available, it becomes simultaneously TRUE with "CommandAborted". In this case, the function block has to be called until "Shutdown" becomes FALSE
	O	Error	If "Error" is TRUE, the function block has been interrupted due to an error and is now in a final state. "Error" remains TRUE until the control input "Enable" is disabled. An additional reset input is not necessary. If a "Shutdown" output is available, it becomes simultaneously TRUE with "Error". In this case, the function block has to be called until "Shutdown" becomes FALSE

Tab. 2-5: I/O of state-controlled function blocks, final processing



If it is required that certain inputs are not only applied at the edge at "Enable", but cyclically during the function block processing, it is to be documented explicitly and not to be specified via the variable name!



For function blocks with "Shutdown", define the behavior of the function block when "Enable" is set again to TRUE while "Shutdown" remains set (retrigger).

Standard interfaces at function blocks

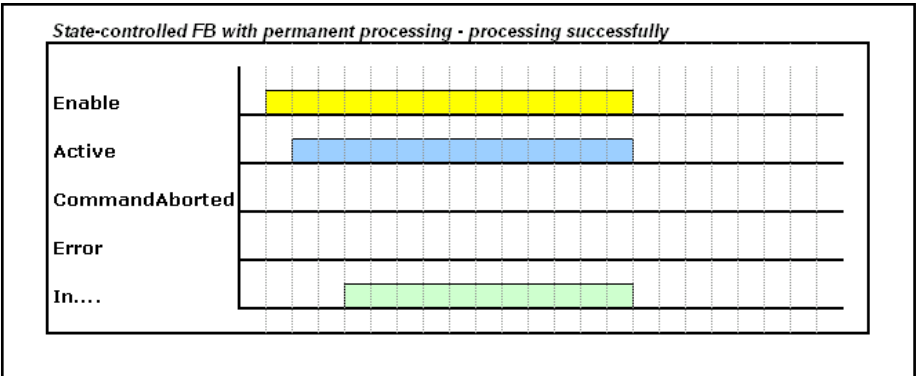


Fig. 2-13: I/O of state-controlled function blocks, permanent processing without "Shutdown". Processing successful

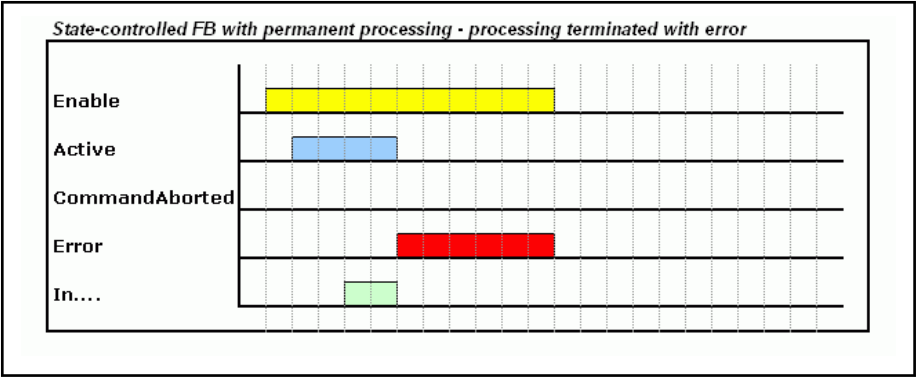


Fig. 2-14: I/O of state-controlled function blocks, permanent processing without "Shutdown". Processing aborted with errors

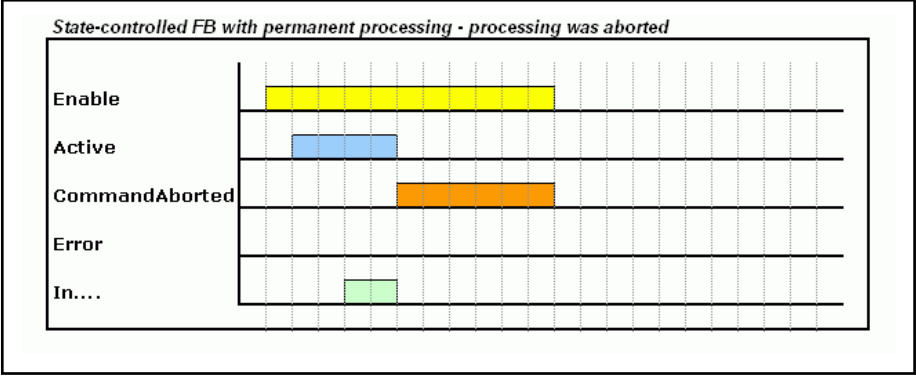


Fig. 2-15: I/O of state-controlled function blocks, permanent processing without "Shutdown". Processing interrupted

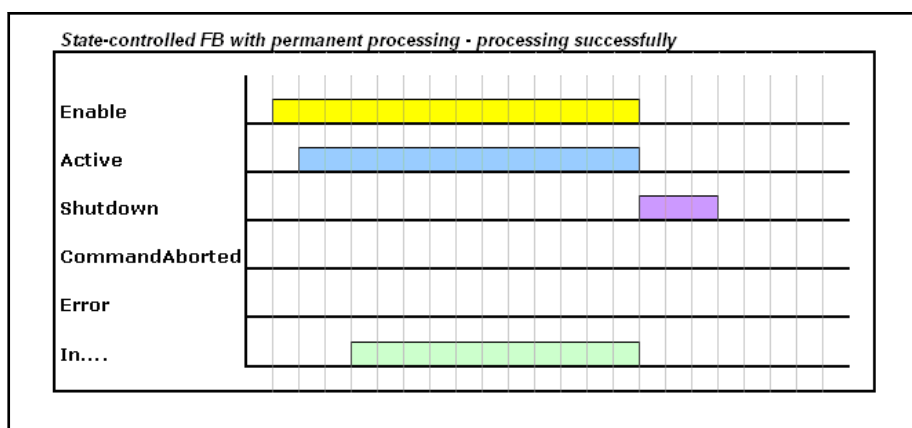


Fig. 2-16: I/O of state-controlled function blocks, permanent processing with "Shutdown". Processing successful

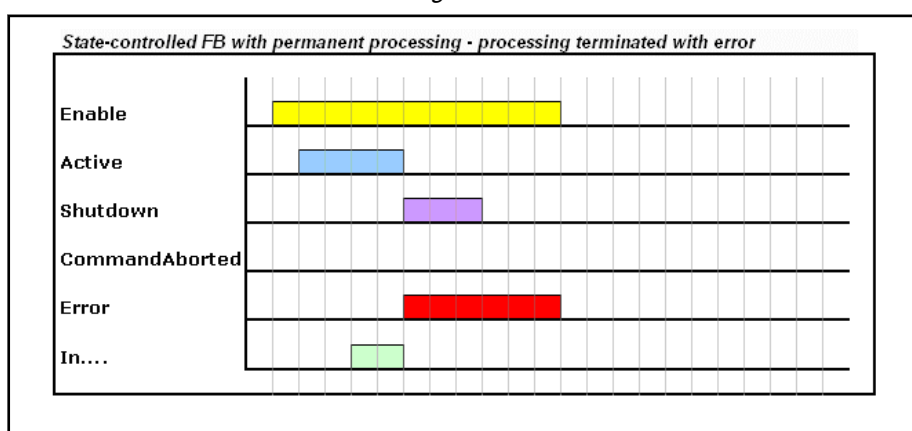


Fig. 2-17: I/O of state-controlled function blocks, permanent processing with "Shutdown". Processing aborted with errors

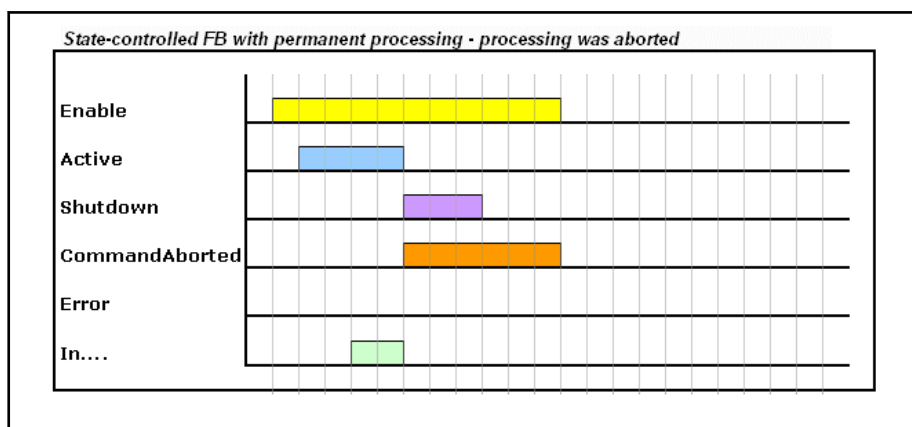


Fig. 2-18: I/O of state-controlled function blocks, permanent processing with "Shutdown". Processing interrupted

2.7 Inputs and outputs of edge-triggered function blocks, permanent processing

Control/ processing	I/O	Variable name	Description
Edge-controlled/ permanent	i	Execute	The input variables are registered with a positive edge of "Execute". New input values are only applied at another positive edge at "Execute". The change of edges at "Execute" suffices to start the block. For further processing, the state of "Execute" is not important. By changing the edge again during processing, the previous task is discarded, the inputs are applied again and the task is continued with the new values (retriggering, subsequent triggering). If the function block provided with a "Shutdown" output, retriggering could not be allowed
	O	In....	If "In...." is TRUE, the function block reached its goal but is still in intervention to "retain" the achieved and it is thus in a permanent final state. Data outputs are valid now. Since the function block remains in intervention, "Active" also has to remain TRUE. This even applies when "In...." decreases again. "Error" and "CommandAborted" are FALSE! If the control input "Execute" is FALSE when the order is completed, "In...." remains TRUE for exactly one PLC cycle. If "Execute" is TRUE, "In...." remains TRUE until "Execute" is disabled
	O	Active	If "Active" is TRUE, the function block processes its actual task. Possible pre-processing are not marked with this output! Since the function block remains in intervention permanently, "Active" remains TRUE until the block is completed by "Error" or "CommandAborted". As long as "Active" is TRUE, "Error" or "CommandAborted" as well as "Shutdown" has to be FALSE!
	O	Shutdown (optional)	If a "Shutdown" output is available, the function block requires multiple cycles of computation time to reach a defined final state and to release the resources used after the function block has canceled its task ("CommandAborted" or "Error" = TRUE). The function block has to be called until "Shutdown" becomes FALSE!
	O	CommandAborted	If "CommandAborted" is TRUE, the function block has been interrupted and is now in a final state. "In....", "Active" and "Error" are FALSE! If the control input "Execute" is FALSE at the time when the order is interrupted, "CommandAborted" remains TRUE for exactly one PLC cycle. If "Execute" is TRUE, "CommandAborted" remains TRUE until "Execute" is disabled. If a "Shutdown" output is available, it becomes simultaneously TRUE with "CommandAborted". In this case, the function block has to be called until "Shutdown" becomes FALSE
	O	Error	If "Error" is TRUE, the function block was interrupted due to an error and is now in a final state. If the control input "Execute" is FALSE when the error occurs, "Error" remains TRUE for exactly one PLC cycle. If "Execute" is TRUE, "Error" remains TRUE until "Execute" is disabled. If a "Shutdown" output is available, it becomes simultaneously TRUE with "Error". In this case, the function block has to be called until "Shutdown" becomes FALSE

Tab. 2-6: I/O of edge-controlled function blocks, permanent processing



If certain inputs are not only applied with the 0/1 edge at "Execute", but cyclically during the function block processing, it is to be documented explicitly and not to be specified via the variable name!



For function blocks with "Shutdown", define the behavior of the function block when a new edge is detected at "Execute" while "Shutdown" remains set (retrigger).

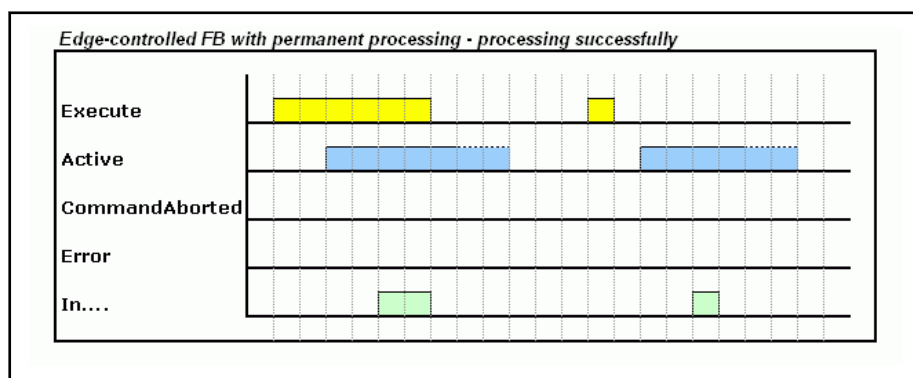


Fig. 2-19: I/O of edge-controlled function blocks, permanent processing without "Shutdown". Processing successful

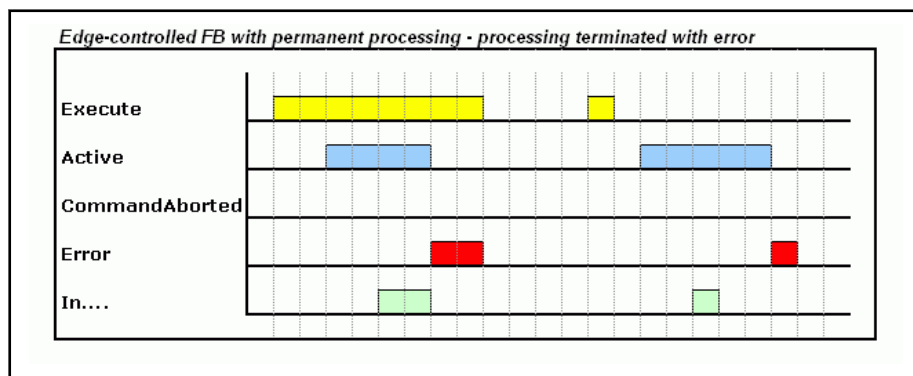


Fig. 2-20: I/O of edge-controlled function blocks, permanent processing without "Shutdown". Processing aborted with errors

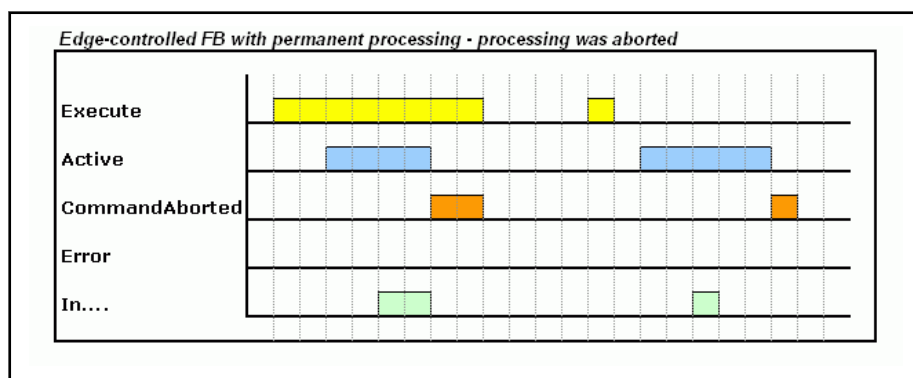


Fig. 2-21: I/O of edge-controlled function blocks, permanent processing without "Shutdown". Processing interrupted

Standard interfaces at function blocks

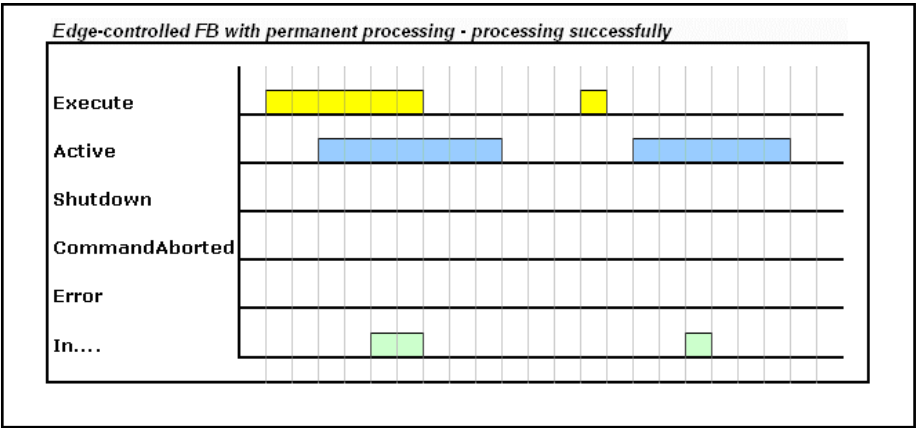


Fig. 2-22: I/O of edge-controlled function blocks, permanent processing with "Shutdown". Processing successful

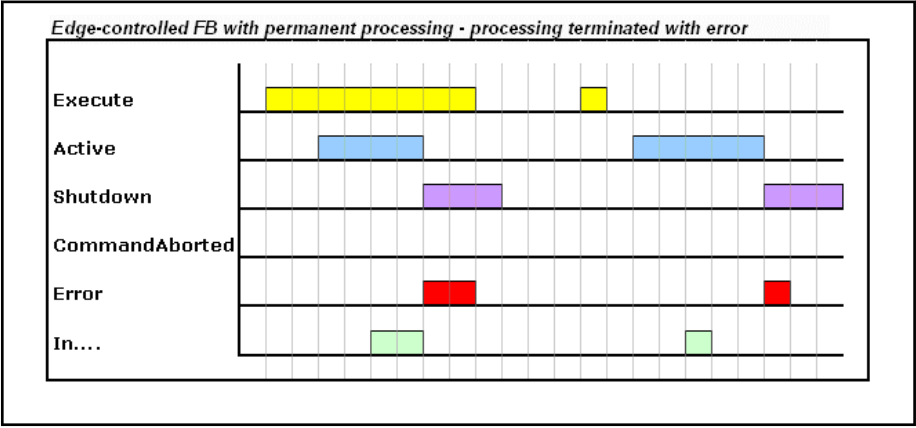


Fig. 2-23: I/O of edge-controlled function blocks, permanent processing with "Shutdown". Processing aborted with errors

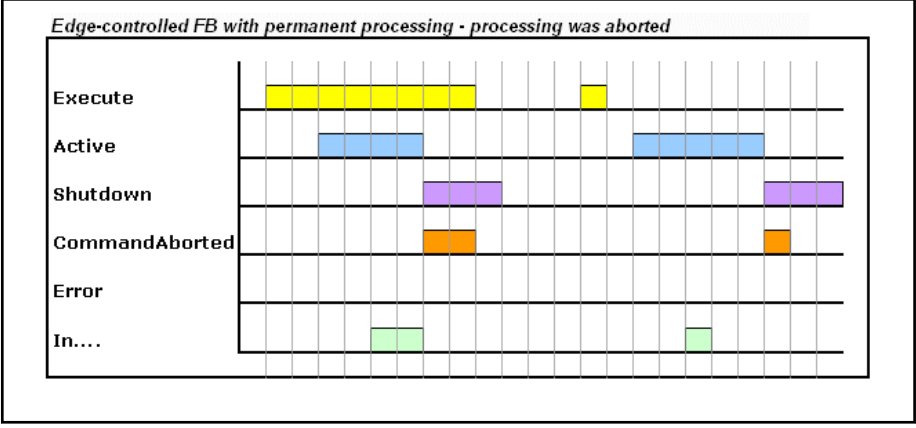


Fig. 2-24: I/O of edge-controlled function blocks, permanent processing with "Shutdown". Processing interrupted

3 RIL_CommonTypes.library

3.1 Introduction and overview



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

The RIL_CommonTypes library contains the data types which describe the input and output variables of the Bosch Rexroth function blocks.

Library structure:

Global_Variables

Global_Variables

- [IL_NULL](#), page 20, NIL pointer.

SercosAttribute

- The Global Variable List contains the Sercos attributes, including a comment on their use.

Data types

Data types of logical addresses

- [CONTROLS](#), page 21, (ENUM)
- [KIN_OBJECTS](#), page 24, (ENUM)
- [OBJECTS](#), page 22, (ENUM)
- [TP_OBJECTS](#), page 25, (ENUM)

PLCopen data types

- [AXIS_GROUP_REF](#), page 23, (STRUCT),
- [AXIS_REF](#), page 21, (STRUCT)
- [MC_CAM_ID](#), page 26, (ENUM),
- [MC_START_MODE](#), page 51, (ENUM)
- [MC_STEP_MODE](#), [MotionProfile extended to FlexProfile](#) , page 26, (ENUM)
- [MC_SYNC_DIRECTION](#), page 53, (ENUM)
- [TOUCHPROBE_REF](#), page 25, (STRUCT)

Data types of POU diagnostics

- [ERROR_STRUCT](#), page 55, (STRUCT)

Includes

- [ERROR_CODE](#), page 54, (ENUM)
- [ERROR_TABLE](#), page 56, (ENUM)

Time data types

- [EXT_SYS_TIME_DATE](#), page 57, (STRUCT)
- [SYS_TIME64](#), page 57, (STRUCT)
- [SYS_TIME_DATE](#), page 58, (STRUCT)

TouchProbe

- [MB_DIFF_EVAL](#), page 59, (ENUM)
- [MB_EXPECT_WINDOW_MODE](#), page 60, (ENUM)
- [MB_INTERPOLATION_MODE](#), page 60, (ENUM)
- [MB_POSITION_EVAL](#), page 61, (ENUM)
- [MB_PROBE_DATA](#), page 62, (ENUM)
- [MB_PROBE_INPUT](#), page 61, (ENUM)
- [MB_PROBE_MODULE](#), page 61, (ENUM)
- [MB_PROBE_CONFIG](#), page 63, (ENUM)

It was declared to define a uniform access to different device types for different control systems.

3.2 Global_Variables

3.2.1 IL_NULL

Brief description A NIL pointer was defined as POINTER TO BYTE.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/Global_Variables
PLC basic package	

Tab. 3-1: *IL_NULL* reference table

Type definition

```
VAR_GLOBAL CONSTANT
  IL_NULL: POINTER TO BYTE := 0;
END_VAR
```

3.3 Data types to reference objects of Motion function blocks

3.3.1 Referencing objects, introduction and overview

To reference objects (controls, axes, kinematics/axis groups, touch probes, etc.), one REF structure with allocated enumeration types is provided for each object type.

Information is transferred to the respective function block via **VAR_IN_OUT CONSTANT** for 14VRS and higher.

The REF structure contains two 16 bit address elements:

- Control address (control address element (element of [CONTROLS](#), page 21) and
- Object address, element of enumeration type



Information was transferred up to 13VRS via **VAR_IN_OUT**.

In 14VRS and higher, information is transferred via **VAR_IN_OUT CONSTANT**.

REF structure	Enumeration type
AXIS_REF , page 21, axes, currently still programmable limit switches	OBJECTS , page 22

AXIS_GROUP_REF , page 23, axis groups/kinematics	KIN_OBJECTS , page 24
TOUCHPROBE_REF , page 25, touch probe	TP_OBJECTS , page 25

Tab. 3-2: REF structures and their enumeration types



The elements of the enumeration types used for the actual individual case depend on

- System
- Control type/drive type
- Firmware version

3.3.2 CONTROLS

Brief description

The enumeration contains all controls addressed via Motion function blocks in the link:



In case of a physical address, there are two options to go to the "local control": number "0", locally, and via its own address, e.g. CTRL_3.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of logical addresses

Tab. 3-3: CONTROLS reference table

Type definition

```
// Enumeration of valid logical control addresses
TYPE CONTROLS :
(
    LOCAL_CNTRL := 0,      // Local control (default)
    CNTRL_1 := 1,         // Control 1
    CNTRL_2 := 2,         // Control 2
    // ...
    CNTRL_99 := 99        // Control 99
);
END_TYPE
```

Functional description

This list of connected controls contains all controls that can be addressed via Motion function blocks in the link:

- Number 0 is enabled for the local control (control itself),
- The number range 1... 99 is reserved for all connected controls.

3.3.3 AXIS_REF

Brief description

The structure contains information on the desired control and on the object controlled by this control (axis, programmable limit switch, oscilloscope,...).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of PLCopen

Tab. 3-4: AXIS_REF reference table

Type definition

```

(* Structure of PLCopen element AXIS_REF. The AXIS_REF is a structure that contains *)
(* information on the corresponding Object. *)
(* It is used as a VAR_IN_OUT CONSTANT in all Motion Control Function Blocks. *)
TYPE AXIS_REF :
STRUCT
  CntrlNo: CONTROLS := LOCAL_CNTRL; (* control reference number *)
                                     (* Steuerungsnummer, Init: lokale Steuerung *)
  AxisNo: OBJECTS := NO_OBJECT; (* object reference number *)
                                   (* Achsreferenznummer, Init: kein Objekt *)
END_STRUCT
END_TYPE

```

Functional description Information is transferred to the respective PLCopen function block via **VAR_IN_OUT CONSTANT** for 14VRS and higher.

The AXIS_REF structure contains two 16 bit address elements:

- Control address (element of [CONTROLS](#), page 21)
- Object address (element of [OBJECTS](#), page 22)

The elements can be addressed via logical or physical address. In case of a logical address, the element points to a parameter structure with a physical address (e.g. Ethernet address of the control, Sercos address of the drive or similar).



Information was transferred up to 13VRS via **VAR_IN_OUT**.

In 14VRS and higher, information is transferred via **VAR_IN_OUT CONSTANT**.

3.3.4 OBJECTS

Brief description This enumeration of available axes contains all the objects of a link addressed via "AXIS_REF" using Motion function blocks.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of logical addresses

Tab. 3-5: *OBJECTS reference table*

Type definition

```

// Enumeration of valid logical object addresses
TYPE OBJECTS :
(
  // not specified object
  NO_OBJECT := 0, // no object specified
  // drives 0001..0999
  AXIS_1 := 1, // drive 1
  // ...
  AXIS_99 := 99, // drive 99
  AXIS_100 := 100, // drive 100
  // ...
  AXIS_512 := 512, // drive 512

  RMA_1 := 1001, (* real master axis 1 *)
  (* RMA_2 := 1002, (* real master axis 2 *)

  VMA_1 := 2001, (* virtual master axis 1 *)
  VMA_2 := 2002, (* virtual master axis 2 *)
  VMA_INT := 2003, (* internal virtual master axis *)
  VMA_EXT := 2004, (* external virtual master axis *)
  MA_LINK_1 := 2005, (* link axis 1 *)
  MA_LINK_2 := 2006, (* link axis 2 *)

  // ELS Group Local Master 3001..3999
  GLMA_1 := 3001, // ELS Group Local Master Axis 1
  // ...

```

```

    GLMA_8 := 3008,           // ELS Group Local Master Axis 8
// ELS Groups 4001..4008
    ELSG_1 := 4001,           // ELS Group 1
    // ...
    ELSG_8 := 4008,           // ELS Group 8
// ELS System Masters 5001..5006
    ELSM_1 := 5001,           // ELS System Master 1
    // ...
    ELSM_6 := 5006,           // ELS System Master 6
// FieldBus Drives 6001..6099
    FieldBusAxis_1 := 6001,    //Fieldbus Drive 1
    // ...
    FieldBusAxis_99 := 6099,    //Fieldbus Drive 99
// Program Limit Switch 8001..8999
    PLS_1 := 8001,             //Program Limit Switch unit 1
    // ...
    PLS_4 := 8004,             //Program Limit Switch unit 4
// Oscilloscope 9001..9999
    OSCI_CHANNEL_1 :=9001,      // Oscilloscope Channel 1
    // ...
    OSCI_CHANNEL_128 :=9128     // Oscilloscope Channel 128
// Sercos address 10001..10512
    SERCOS_ADDRESS_1 :=10001,    // Sercos address 1
    // ...
    SERCOS_ADDRESS_512 :=10512,  // Sercos address 512
// Sercos topology 11001..11512
    SERCOS_TOPOLOGY_1 :=11001,   // Sercos topology 1
    // ...
    SERCOS_TOPOLOGY_512 :=11512, // Sercos topology 512
// Logical drive 12001..12512 with native operation data *)
    AXIS_NATIVE_1 := 12001,      // Logical drive 1 with native operation data
    // ...
    AXIS_NATIVE_512 := 12512     // Logical drive 512 with native operation data
);
END_TYPE

```



Number "00", "NO_OBJECT" (no object specified) is used for de-activated axes.



Name and numbers are assigned when the respective axis/the object is created in the IndraWorks Suite (IndraLogic, Global-MlcVariables).

3.3.5 AXIS_GROUP_REF

Brief description The structure contains information on the desired control and the corresponding axis group.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of PLCopen

Tab. 3-6: *AXIS_GROUP_REF* reference table

Type definition

```

// Structure of PLCopen element AXIS_GROUP_REF.
// The AXIS_GROUP_REF is a structure that contains information of the axis-group.
// It is used as a VAR_IN_OUT CONSTANT in Motion Control Function Blocks.
TYPE AXIS_GROUP_REF:
STRUCT
    CntrlNo: CONTROLS := LOCAL_CNTRL; // control reference number
                                         // Steuerungsnummer, Init: lokale Steuerung

```

RIL_CommonTypes.library

```

GroupNo: KIN_OBJECTS := NO_KINEMATICS; // object reference number
// Achsgruppenreferenznummer, Init: NO_KINEMATICS
END_STRUCT
END_TYPE

```

Functional description

Information is transferred to the respective function block via **VAR_IN_OUT CONSTANT** for 14VRS and higher.

The **AXIS_GROUP_REF** structure contains two 16 bit address elements:

- Control address (element of [CONTROLS](#), page 21)
- Object address (element of [KIN_OBJECTS](#), page 24)

The elements can be addressed via logical or physical address. In case of a logical address, the element points to a parameter structure with a physical address (e.g. Ethernet address of the control).



Information was transferred up to 13VRS via **VAR_IN_OUT**.

In 14VRS and higher, information is transferred via **VAR_IN_OUT CONSTANT**.

3.3.6 KIN_OBJECTS

Brief description

The enumeration of available kinematics (axis groups) contains all objects of a Motion link accessed by Coordinated Motion function blocks, also refer to ([AXIS_GROUP_REF](#), page 23).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of logical addresses

Tab. 3-7: *KIN_OBJECTS reference table*

Type definition

```

TYPE KIN_OBJECTS :
(
  (* not specified object
    NO_KINEMATICS :=0,
  (* axis groups 1..16 *)
    KINEMATICS_1 := 01,
    KINEMATICS_2 := 02,
    KINEMATICS_3 := 03,
    KINEMATICS_4 := 04,
    KINEMATICS_5 := 05,
    KINEMATICS_6 := 06,
    KINEMATICS_7 := 07,
    KINEMATICS_8 := 08,
    KINEMATICS_9 := 09,
    KINEMATICS_10 := 10,
    KINEMATICS_11 := 11,
    KINEMATICS_12 := 12,
    KINEMATICS_13 := 13,
    KINEMATICS_14 := 14,
    KINEMATICS_15 := 15,
    KINEMATICS_16 := 16
  );
END_TYPE

```



Name and numbers are assigned when the object is created in the IndraWorks Suite (IndraLogic, GlobalMlcVariables)



The number of implementable kinematics depends on the control type and the firmware version.

3.3.7 TOUCHPROBE_REF

Brief description The structure contains information on the desired control and on the object controlled by this control (touch probe).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of PLCopen

Tab. 3-8: TOUCHPROBE_REF reference table

Type definition

```
(* Structure of PLCopen element TOUCHPROBE_REF. The TOUCHPROBE_REF is a structure that *)
(* contains information on the corresponding object. *)

TYPE TOUCHPROBE_REF :
STRUCT
  CntrlNo:      CONTROLS      := LOCAL_CNTRL;  (* control reference number *)
              (* Steuerungsnummer, Init: lokale Steuerung *)
  TouchProbeNo: TP_OBJECTS := NO_TOUCHPROBE  (* object reference number *)
              (* Messtastereferenznummer, Init: kein Objekt *)
END_STRUCT
END_TYPE
```

Functional description Information is transferred to the respective PLCopen function block via **VAR_IN_OUT CONSTANT** for 14VRS and higher.

The TOUCHPROBE_REF structure contains two 16 bit address elements:

- Control address (element of [CONTROLS](#), page 21)
- Object address (element of [TP_OBJECTS](#), page 25).



Information was transferred up to 13VRS via **VAR_IN_OUT**.

In 14VRS and higher, information is transferred via **VAR_IN_OUT CONSTANT**.

3.3.8 TP_OBJECTS

Brief description The enumeration of available touch probes contains all objects of a Motion link accessed by touch probe function blocks, also refer to ([TOUCHPROBE_REF](#), page 25).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of logical addresses

Tab. 3-9: TP_OBJECTS reference table

Type definition

```
(* Enumeration of valid logical object addresses *)
TYPE TP_OBJECTS :
(
  (* not specified object *)
  NO_TOUCHPROBE := 0,      (* no object specified *)
  (* Touch Probe 1..100, realized on controller *)
  TOUCHPROBE_1  := 01,    (* Touch Probe 1 *)
  (* ... *)
  TOUCHPROBE_100 := 100   (* Touch Probe 100 *)
)
```

```

(* Touch Probe 1001..1002, realized on drive      *)
  TOUCHPROBE_DRIVE_1 :=1001  (* Touch Probe Drive 1 *)
  TOUCHPROBE_DRIVE_2 :=1002  (* Touch Probe Drive 2 *)
);
END_TYPE

```



Names and numbers are assigned when the respective touch probe is manually created in the IndraWorks Suite (IndraLogic, GlobalMlcVariables)



The number of implementable touch probes depends on the control type and the firmware version.

3.4 Data types to support cams, MotionProfiles and FlexProfiles

3.4.1 Data type to switch between cams, general information

Depending on the potential of the control electronics, the following cam tables can be stored:

- MPDx - 4 tables with 1,024 points each (table 1...4),
- MPBx - 4 tables with 1,024 points each (table 1...4),
- MPHx - 4 tables with 1,024 points each (table 1...4), 4 tables with 128 points each (table 5...8).

Cam table switching is possible at runtime. Select and switch via the function block MC_CamIn (CamTableID: [MC_CAM_ID](#), page 26) or via the parameter control "A-0-2740, ELS, CamIn, cam table preselection".

3.4.2 MC_CAM_ID

Brief description The enumeration type supports the selection of the cam table ([Data types to support cams, MotionProfiles and FlexProfiles](#), page 26).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/data types of
PLC basic package	PLCopen

Tab. 3-10: MC_CAM_ID reference table

Type definition

```

TYPE MC_CAM_ID :
(
  CAM_TABLE_1  := 1,  (* Kurventabelle 1 *)
  CAM_TABLE_2  := 2,  (* Kurventabelle 2 *)
  (* ... *)
  CAM_TABLE_255 := 255 (* Kurventabelle 255 *)
);
END_TYPE

```

Functional Description Cam table switching is possible at runtime. Select and switch via the function block MC_CamIn, ML_PLCOpen.library or via the parameter control "A-0-2740, ELS, CamIn, cam table preselection".

3.4.3 MC_STEP_MODE, MotionProfile extended to FlexProfile

Brief description The enumeration data type "MC_STEP_MODE" describes the dependence of the slave axis position from the master axis during a motion step (MotionProfile extended to FlexProfile).

The dependence can be given as functional dependence or as cam table.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of PLCopen

Tab. 3-11: MC_STEP_MODE reference table

Type definition

```

TYPE MC_STEP_MODE :
(
    REST_IN_REST_INCLINEDSINE          := 16#0000, (* MotionProfile, FlexProfile *)
    REST_IN_REST_POLY5                  := 16#0100, (* MotionProfile, FlexProfile *)
    REST_IN_VELOCITY_POLY5              := 16#0200, (* MotionProfile, FlexProfile *)
    REST_IN_VELOCITY_POLY7              := 16#0300, (* FlexProfile *)
    VELOCITY_IN_REST_POLY5              := 16#0400, (* MotionProfile, FlexProfile *)
    VELOCITY_IN_REST_POLY7              := 16#0500, (* FlexProfile *)
    CONSTANT_VELOCITY                   := 16#0600, (* MotionProfile, FlexProfile *)
    VELOCITY_IN_VELOCITY_POLY5          := 16#0700, (* MotionProfile, FlexProfile *)

    REST_IN_REST_POLY5_VLIM              := 16#1200, (* MotionProfile, FlexProfile *)
    REST_IN_REST_PARABOLA                := 16#1300, (* FlexProfile *)
    REST_IN_REST_POLY8                  := 16#1400, (* FlexProfile *)

    REST_IN_REST_LINEAR                  := 16#0800, (* FlexProfile *)
    REST_IN_REST_POLY7                  := 16#0900, (* FlexProfile *)
    REST_IN_REST_SINE                    := 16#0A00, (* FlexProfile *)
    REST_IN_REST_GUTMANSINE              := 16#0B00, (* FlexProfile *)
    REST_IN_REST_SINEACC                 := 16#0C00, (* FlexProfile *)
    REST_IN_REST_SINETORQUE              := 16#0D00, (* FlexProfile *)
    REST_IN_REST_MOD_TRAPEZE             := 16#0E00, (* FlexProfile *)
    REST_IN_REST_MOD_SINE                := 16#0F00, (* FlexProfile *)

    VELOCITY_IN_VELOCITY_POLY7           := 16#1000, (* FlexProfile *)
    VELOCITY_IN_VELOCITY_MOD_SINE        := 16#1100, (* FlexProfile *)

    MOTION_IN_MOTION_POLY5               := 16#2000, (* FlexProfile, *)
    MOTION_IN_MOTION_POLY7               := 16#2100, (* FlexProfile, *)
    MOTION_IN_MOTION_POLY7               (* auch FlexStep / also Flex motion step *)

    COMMON_POLY5                         := 16#4000, (* FlexProfile *)
    COMMON_POLY7                         := 16#4100, (* FlexProfile *)
    COMMON_POLY2                         := 16#4200, (* FlexProfile *)
    COMMON_POLY3                         := 16#4300, (* FlexProfile *)
    COMMON_POLY4                         := 16#4400, (* FlexProfile *)
    COMMON_POLY8                         := 16#4500, (* FlexProfile *)

    (* Erweiterte Bewegungsgesetze / Extended motion laws *)
    X_VELOCITY_IN_VELOCITY_TRAPEZE_ALIM := 16#5000, (* FlexProfile *)
    X_MOTION_IN_MOTION_POLY5_VLIM      := 16#6000, (* FlexProfile *)
    X_MOTION_IN_MOTION_POLY5_SLIM      := 16#6100, (* FlexProfile, *)
    X_MOTION_IN_MOTION_POLY5_SLIM      (* Auch FlexStep / also Flex motion step *)

    X_FIT_VEL_TRAPEZE_ALIM               := 16#7000, (* FlexProfile *)
    X_FIT_SINE_TRAPEZE_ALIM              := 16#7100, (* FlexProfile *)
    X_FIT_ACC_TRAPEZE_ULIM               := 16#7200, (* FlexProfile *)

    X_MOTION_IN_MOTION_ACAM              := 16#8000, (* FlexProfile, *)
    X_MOTION_IN_MOTION_VCAM              := 16#8100, (* FlexProfile, *)
    X_MOTION_IN_MOTION_VCAM2_A          := 16#8200, (* FlexProfile, *)
    X_MOTION_IN_MOTION_VCAM2_B          := 16#8300, (* FlexProfile, *)

    (* Punktetabellen MotionProfile / Cam tables *)
    CAMTABLE_1                          := 1,        (* C-0-2001 *)
    CAMTABLE_2                          := 2,        (* C-0-2002 *)
    CAMTABLE_3                          := 3,        (* C-0-2003 *)
    CAMTABLE_98                         := 98,       (* C-0-2098 *)
    CAMTABLE_99                         := 99,       (* C-0-2099 *)
);
END_TYPE

```

Functional description



The following functional description contains details on Motion-Profiles and FlexProfiles.

Different standard profiles or a cam table can be defined as step mode.

The following modes are available:

- R-R: Rest-in-rest, polynomial 5th order,
- R-R: Rest-in-rest, inclined sinusoid,
- R-V: Rest-in-velocity, polynomial 5th order,
- V-R: Velocity-in-rest, polynomial 5th order,
- G: Constant velocity
- V-V: velocity-in-velocity, polynomial 5th order,
- Cam table.

R-R: Rest-in-rest

For the profile "Rest-in-rest", ten profiles are available. It can be selected between the standardized motion laws "Polynomial 5th order" and "Inclined sine curve" ...

These profiles are defined by the distance and the master axis range. The final velocity of the step is determined by the following step.

The profile for the "**polynomial 5th order**" is determined by the following function:

$$f(z) = 6z^5 - 15z^4 + 10z^3$$

Fig. 3-1: Rest-in-rest, polynomial 5th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-12: Boundary values for the motion law - polynomial 5th order

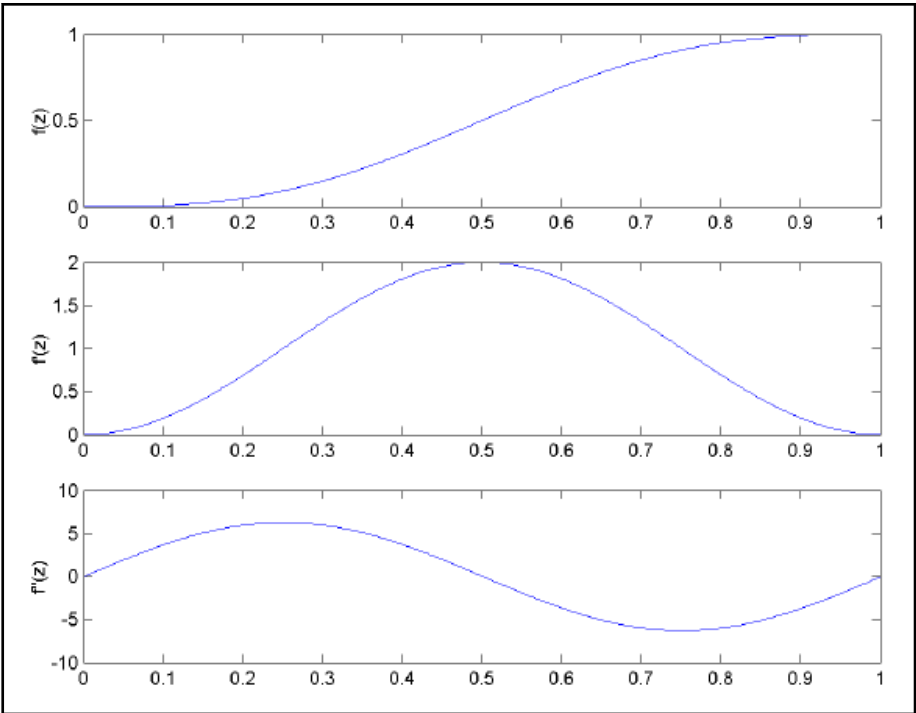


Fig. 3-2: Rest-in-rest, polynomial 5th order

For the "polynomial 7th order", the following formula applies:

$$f(z) = -20z^7 + 70z^6 - 84z^5 + 35z^4$$

Fig. 3-3: Rest-in-rest, polynomial 7th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	0	0

Tab. 3-13: Marginal values for the motion law - Polynomial 7th order

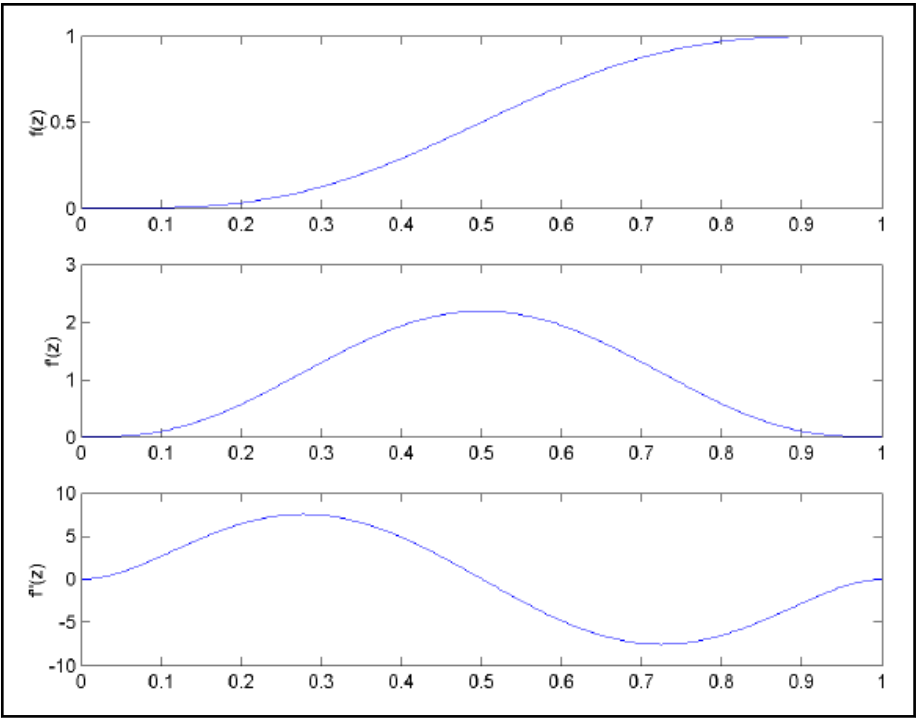


Fig. 3-4: Rest-in-rest, polynomial 7th order

The profile for the **"inclined sine curve"** is determined by the following function:

$$f(z) = z - \frac{1}{2\pi} \sin(2\pi z)$$

Fig. 3-5: Rest-in-rest, inclined sine curve

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-14: Marginal values for the motion law - Inclined sine curve

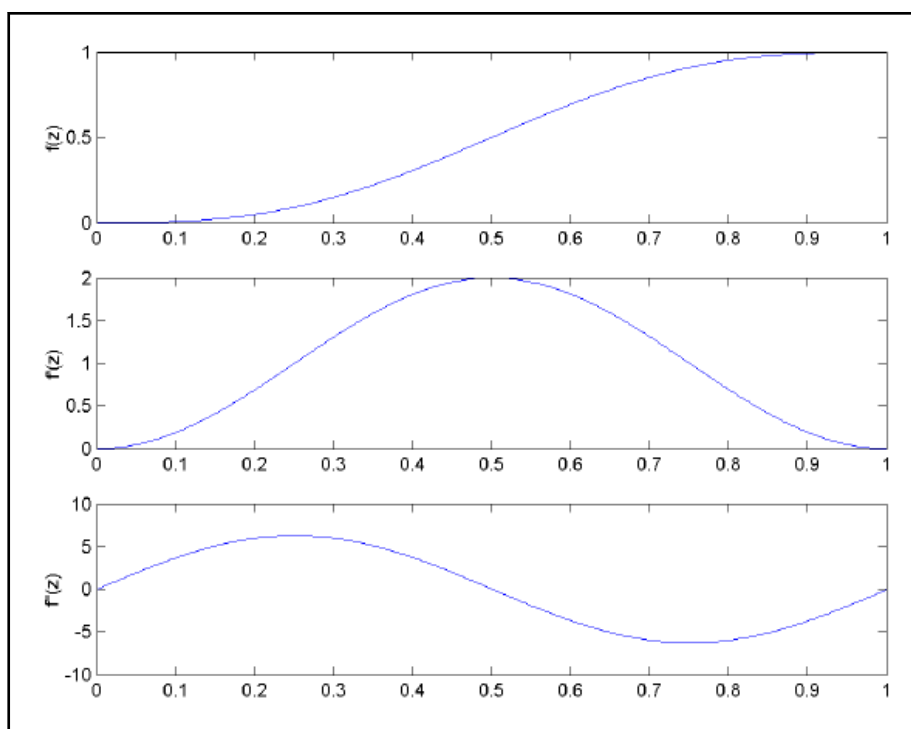


Fig. 3-6: *Rest-in-rest, Inclined sine curve (BESTEHORN)*

The profile for the **"Sine curve (simple sinoid)"** is determined by the following function:

$$f(z) = \frac{1}{2} [1 - \cos(\pi z)]$$

Fig. 3-7: *Rest-in-rest, sine curve*

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	Automatically	Automatically
3. Order (jerk)	Automatically	Automatically

Tab. 3-15: *Marginal values for the motion law "Sine curve"*

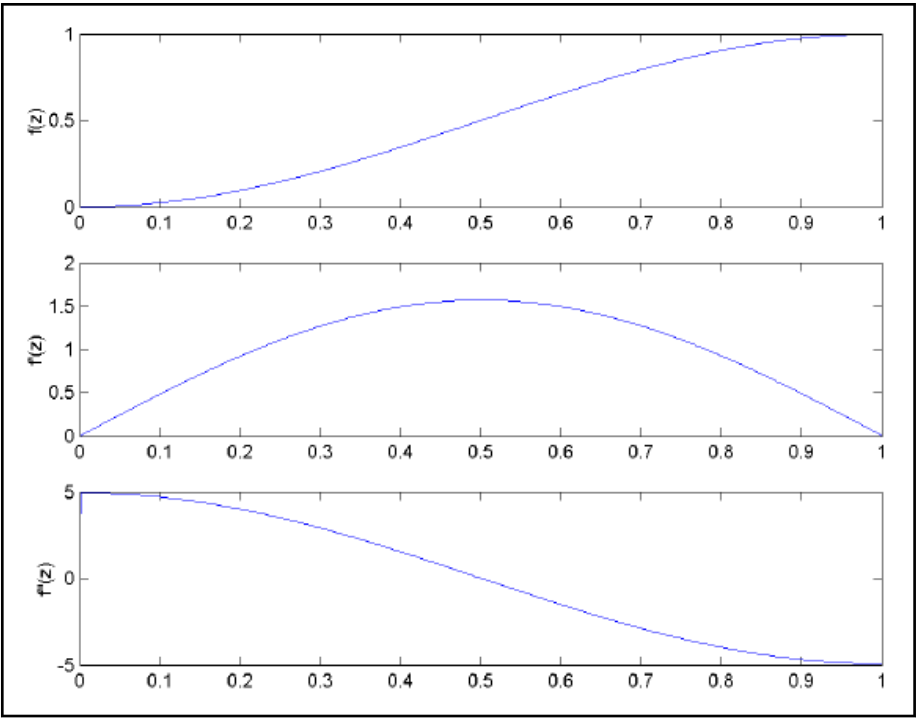


Fig. 3-8: Rest-in-rest, Sine curve (simple sinoid)
The profile for the "Linear rest" is determined by the following function:

$$f(z) = 0$$

Fig. 3-9: Linear rest
The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	0	0

Tab. 3-16: Marginal values for motion laws - Linear rest

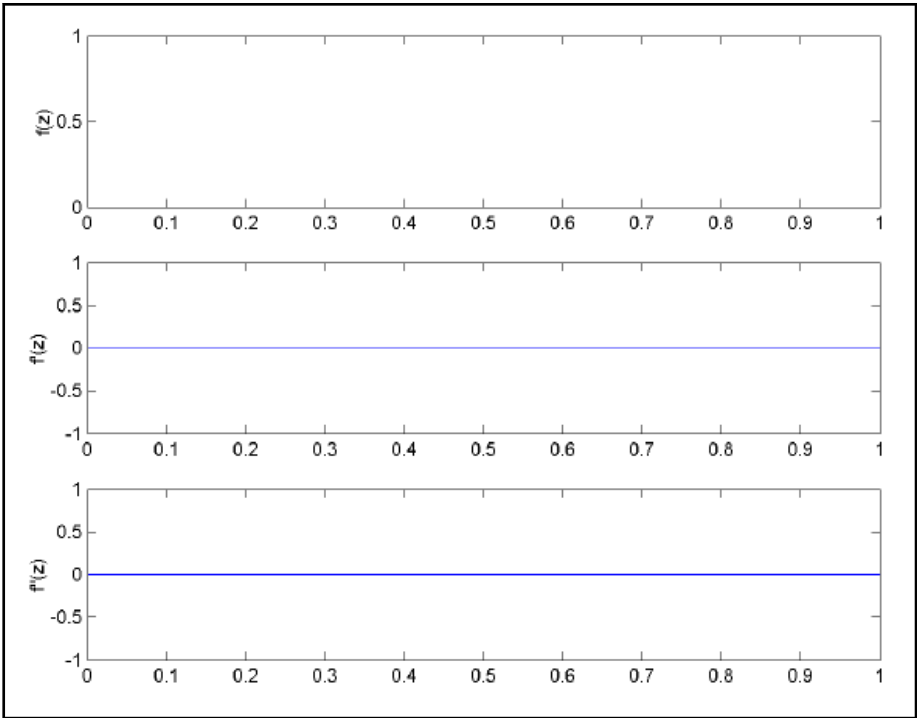


Fig. 3-10: Linear rest

The profile for the "Gutman sinoide" is determined by the following function:

$$f(z) = z - \frac{15}{32\pi} \sin(2\pi z) - \frac{1}{96\pi} \sin(6\pi z)$$

Fig. 3-11: Gutman sinusoid

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-17: Marginal values for the motion law - Gutman sinoide

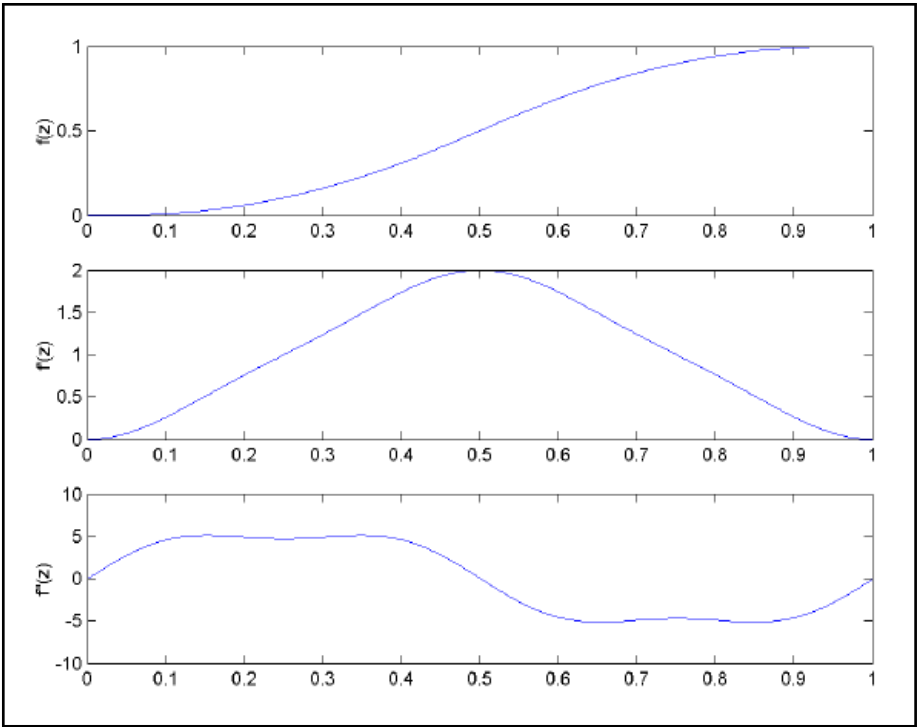


Fig. 3-12: Rest-in-rest, Gutman sinoide

The profile for the **"inclined sine curve optimum for acceleration"** is determined by the following function:

$$f(z) = z - \frac{1}{2\pi} \sin(2\pi z)$$
$$g(z) = z - \frac{0.41}{2\pi} \sin(2\pi z)$$

Fig. 3-13: Acceleration-optimum inclined sinusoid

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-18: Marginal values for the motion law - Inclined sine curve optimum for acceleration

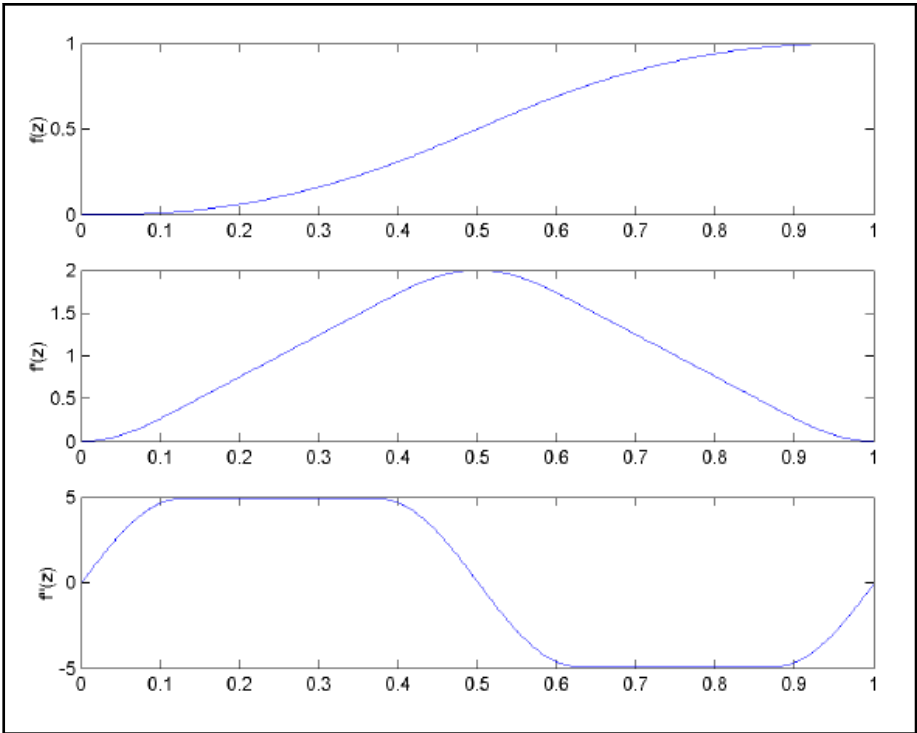


Fig. 3-14: Rest-in-rest, acceleration-optimum inclined sine curve

The profile for the "inclined sine curve optimum for the torque" is determined by the following function:

$$f(z) = z - \frac{1}{2\pi} \sin(2\pi z)$$
$$g(z) = z - \frac{0.134}{2\pi} \sin(2\pi z)$$

Fig. 3-15: Torque-optimum inclined sinusoid

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-19: Marginal values for the motion law - Inclined sine curve optimum for the torque

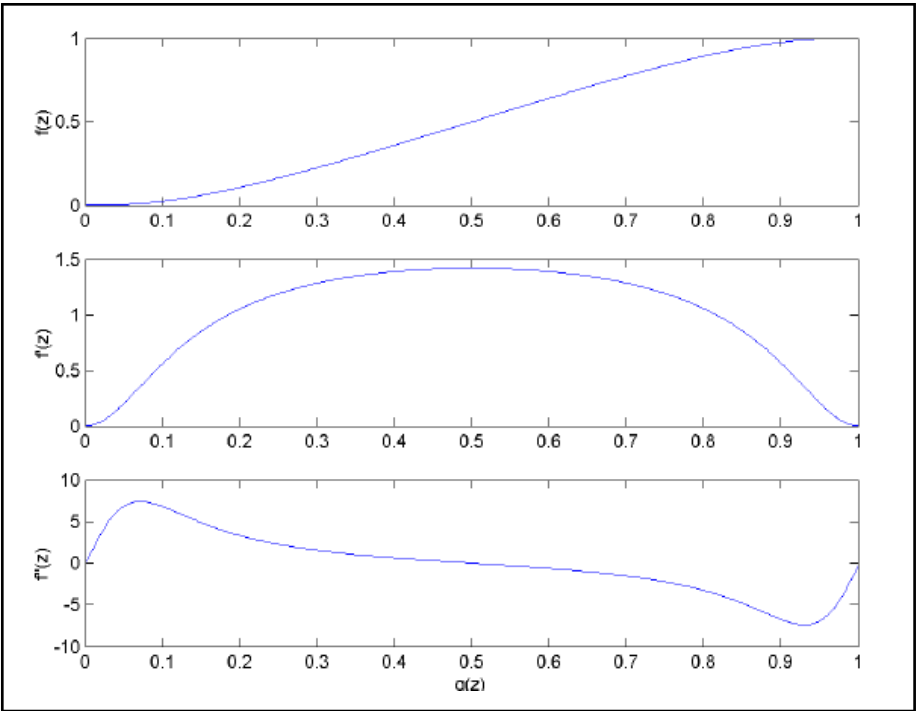


Fig. 3-16: Rest-in-rest, torque-optimum inclined sine curve
The profile for the "Modified sinoide" is determined by the following function:

$$0 \leq z \leq \frac{1}{8} : f(z) = \frac{1}{4 + \pi} \left[\pi z - \frac{1}{4} \sin(4\pi z) \right]$$
$$\frac{1}{8} \leq z \leq \frac{7}{8} : f(z) = \frac{1}{4 + \pi} \left[2 + \pi z - \frac{4}{9} \sin\left(\frac{4\pi}{3} z + \frac{\pi}{3}\right) \right]$$
$$\frac{7}{8} \leq z \leq 1 : f(z) = \frac{1}{4 + \pi} \left[4 + \pi z - \frac{1}{4} \sin(4\pi z) \right]$$

Fig. 3-17: Modified sinusoid
The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-20: Marginal values for the motion law - Modified sinoide

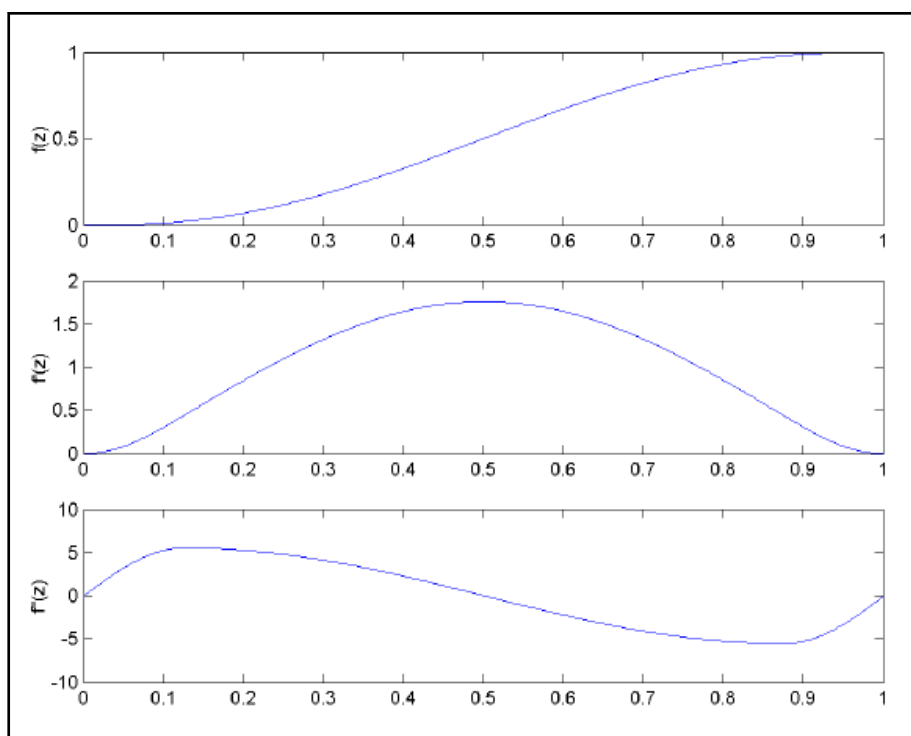


Fig. 3-18: Rest-in-rest, modified sine

The profile for the **"Modified acceleration trapezoid"** is determined by the following function:

$$\begin{aligned}
 0 \leq z \leq \frac{1}{8}: f(z) &= \frac{1}{2+\pi} \left[2z - \frac{1}{2\pi} \sin(4\pi z) \right] \\
 \frac{1}{8} \leq z \leq \frac{3}{8}: f(z) &= \frac{1}{2+\pi} \left[-\frac{1}{2\pi} + 2z + 4\pi \left(z + \frac{1}{8} \right)^2 \right] \\
 \frac{3}{8} \leq z \leq \frac{1}{2}: f(z) &= \frac{1}{2+\pi} \left[-\frac{\pi}{2} + 2(1+\pi)z - \frac{1}{2\pi} \sin(4\pi z - \pi) \right] \\
 \frac{1}{2} \leq z \leq 1: f(z) &= 1 - f(1-z)
 \end{aligned}$$

Fig. 3-19: Modified acceleration trapezoid

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	0
2. Order (acceleration)	0	0
3. Order (jerk)	Automatically	Automatically

Tab. 3-21: Marginal values for the motion law - Modified acceleration trapezoid

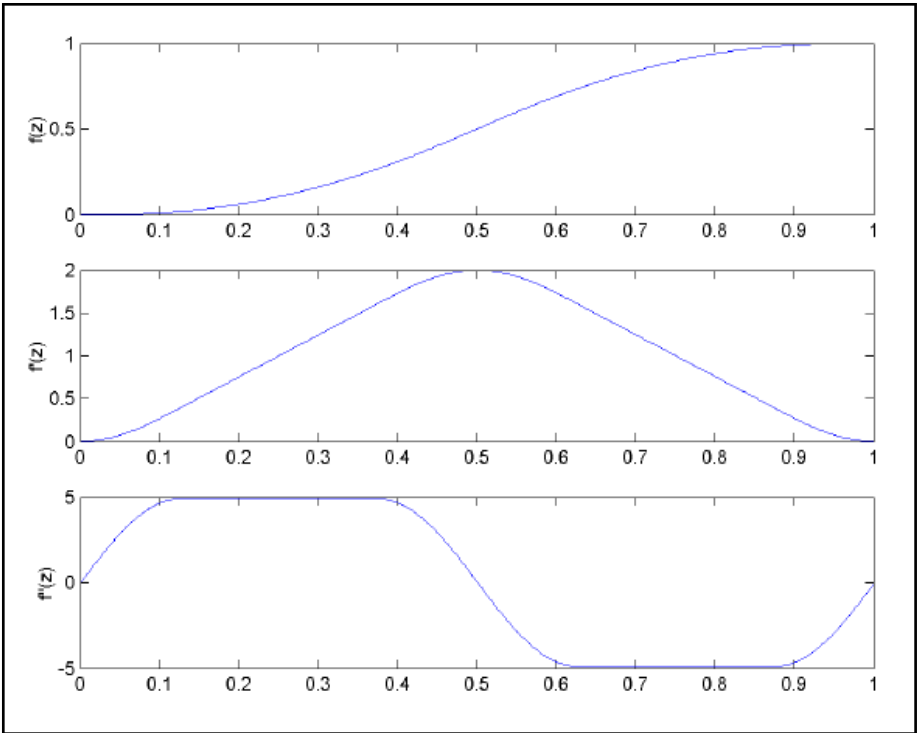


Fig. 3-20: Rest-in-rest, modified acceleration trapezoid

R-V: Rest-in-velocity

This profile is used for the transition from standstill to a certain velocity.
The profile is calculated with a "Polynomial 5th order":

$$f(z) = A_5z^5 + A_4z^4 + A_3z^3$$

Fig. 3-21: Polynomial 5th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	Selectable
2. Order (acceleration)	0	Selectable
3. Order (jerk)	Automatically	Automatically

Tab. 3-22: Marginal values for the motion law - Polynomial 5th order

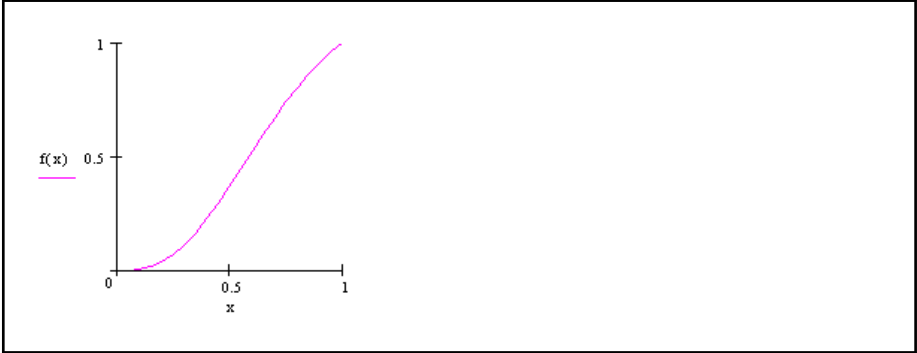


Fig. 3-22: Rest-in-velocity, polynomial 5th order

This profile is used for the transition from standstill to a certain velocity.
The profile is calculated with a "Polynomial 7th order":

$$f(z) = A_7 z^7 + A_6 z^6 + A_5 z^5 + A_4 z^4 + A_3 z^3$$

Fig. 3-23: Polynomial 7th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	0	Selectable
2. Order (acceleration)	0	Selectable
3. Order (jerk)	Automatically	Selectable

Tab. 3-23: Marginal values for the motion law - Polynomial 7th order

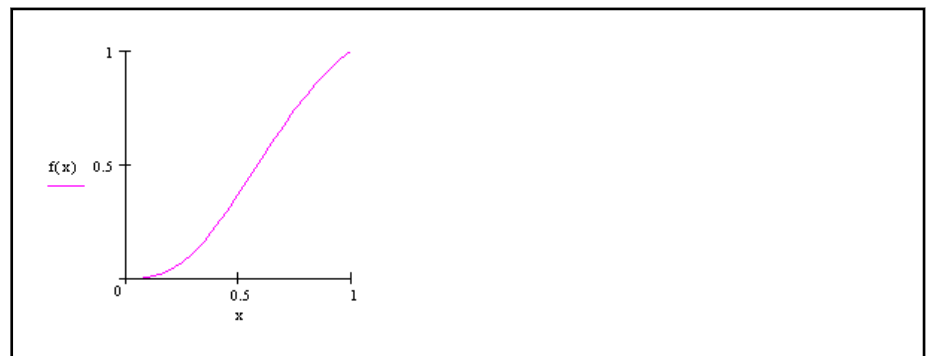


Fig. 3-24: Rest-in-velocity, polynomial 7th order

V-R: Velocity-in-rest

The profile is used for the transition from a defined velocity to the standstill.

The distance traveled in this MotionProfile is defined by the distance for this motion step.

The profile is calculated with a "Polynomial 5th order":

$$f(z) = A_5 z^5 + A_4 z^4 + A_3 z^3 + A_1 z$$

Fig. 3-25: Velocity-in-rest, polynomial 5th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Selectable	0
2. Order (acceleration)	Selectable	0
3. Order (jerk)	Automatically	0

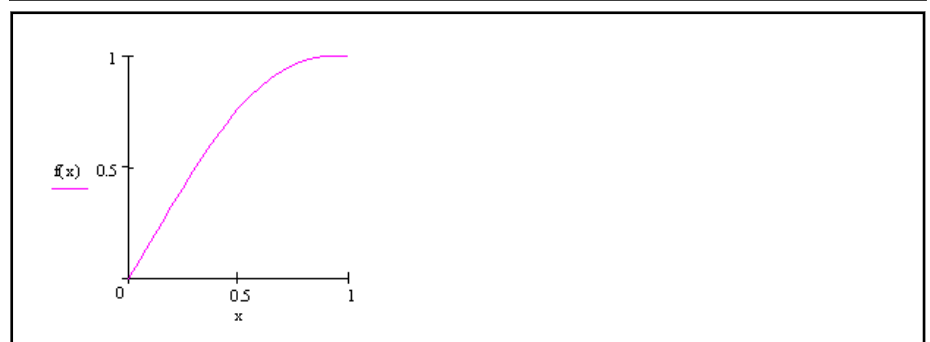


Fig. 3-26: Velocity-in-rest, polynomial 5th order

The profile is used for the transition from a defined velocity to the standstill.

The profile is calculated with a **"Polynomial 7th order"**:

$$f(z) = A_7z^7 + A_6z^6 + A_5z^5 + A_4z^4 + A_3z^3 + A_2z^2 + A_1z$$

Fig. 3-27: Rest-in-velocity, polynomial 7th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Selectable	0
2. Order (acceleration)	Selectable	0
3. Order (jerk)	Selectable	0

Tab. 3-24: Marginal values for the motion law - Polynomial 7th order

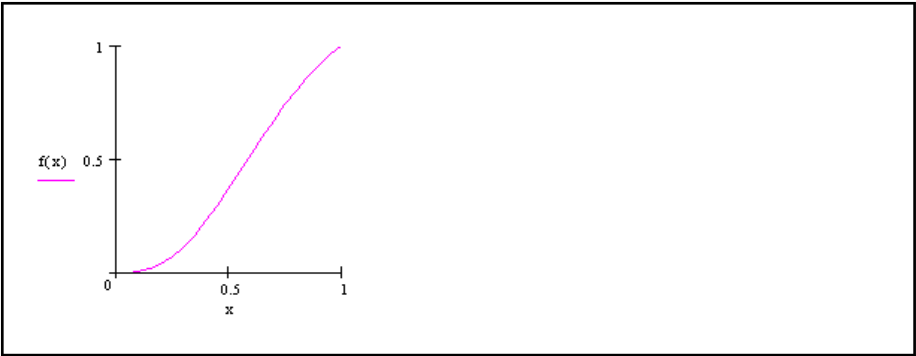


Fig. 3-28: Rest-in-velocity, polynomial 7th order

G: constant velocity

The profile of this motion step is a **"straight line"**. The velocity in this step is constant. The distance traveled is defined by the distance for this motion step.

The formula for the profile is standardized as follows:

$f(z) = z.$

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Automatically	Automatically
2. Order (acceleration)	Automatically	Automatically
3. Order (jerk)	Automatically	Automatically

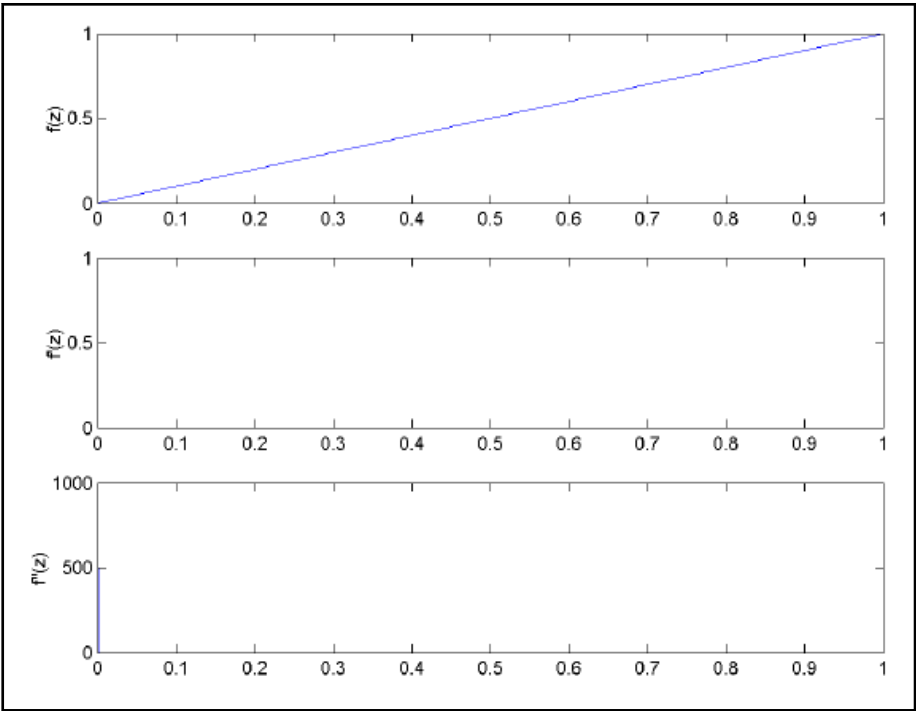


Fig. 3-29: Constant velocity

The velocity of the slave axis is determined from the values specified for this motion step:

- Master axis range
- Distance
- Master axis velocity.

It is not necessary to enter a value for the velocity of the slave axis.

V-V: Velocity-in-velocity The profile for the "polynomial 5th order" is determined by the following function:

$$f(z) = A_5z^5 + A_4z^4 + A_3z^3 + A_2z^2 + A_1z$$

Fig. 3-30: Polynomial 5th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Selectable	Selectable
2. Order (acceleration)	Selectable	Selectable
3. Order (jerk)	Automatically	Automatically

Tab. 3-25: Boundary values for the motion law - polynomial 5th order

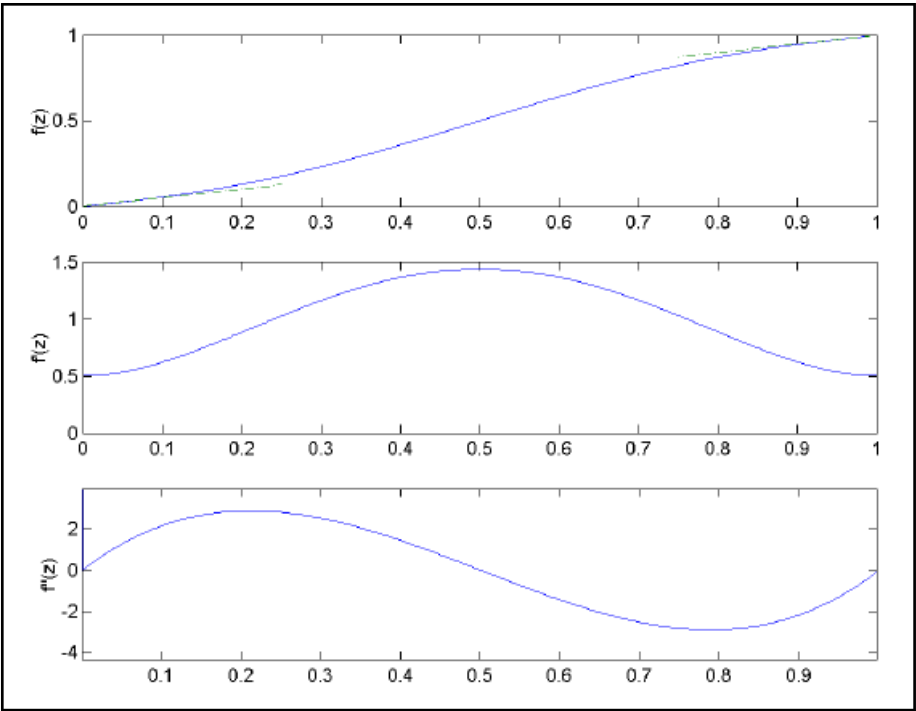


Fig. 3-31: Velocity-in-velocity, polynomial 5th order

The profile for the **"Polynomial 7th order"** is determined by the following function:

$$f(z) = A_7 z^7 + A_6 z^6 + A_5 z^5 + A_4 z^4 + A_3 z^3 + A_2 z^2 + A_1 z$$

Fig. 3-32: Polynomial 7th order

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Selectable	Selectable
2. Order (acceleration)	Selectable	Selectable
3. Order (jerk)	Selectable	Selectable

Tab. 3-26: Marginal values for the motion law - Polynomial 7th order

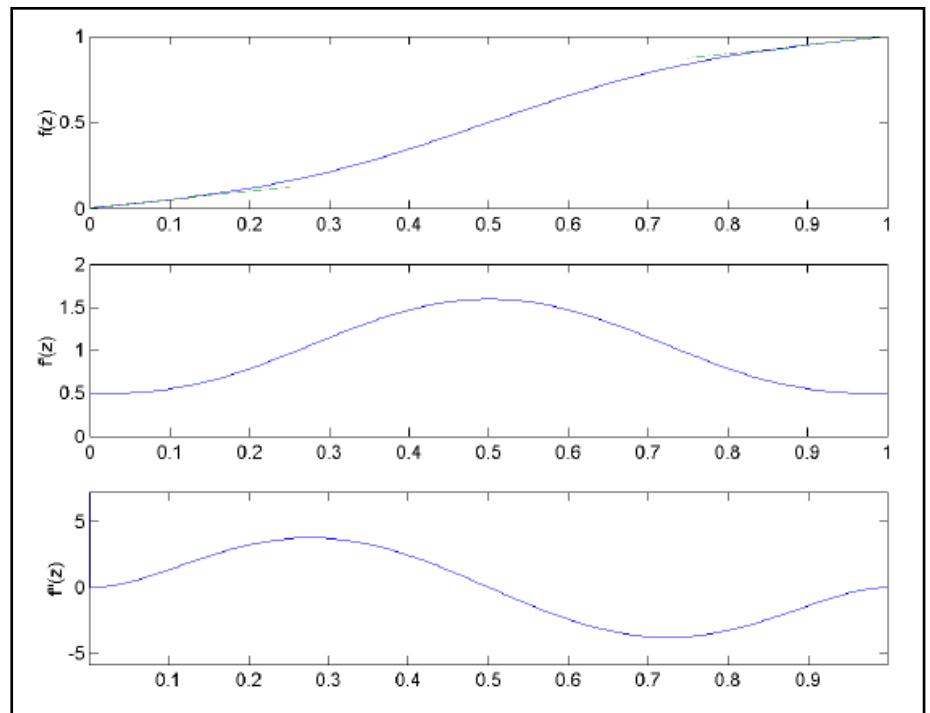


Fig. 3-33: Velocity-in-velocity, polynomial 7th order

The profile for the **"Modified sine curve"** is determined by the following function:

$$\begin{aligned}
 0 \leq z \leq \frac{\lambda}{8}: f'(z) &= C_a \frac{\lambda}{2\pi} \left(z - \frac{\lambda}{2\pi} \sin \left(2\pi \frac{z}{\lambda} \right) \right) + f'(0)z \\
 \frac{\lambda}{8} \leq z \leq \lambda: f'(z) &= C_a \frac{\lambda}{2\pi} \left(z - \frac{9\lambda}{2\pi} \sin \left(2\pi \frac{z + \frac{\lambda}{2}}{3\lambda} \right) + \frac{4\lambda}{\pi} \right) + f'(0)z \\
 \lambda \leq z \leq 1 - \frac{1-\lambda}{4}: f'(z) &= C_b \frac{1-\lambda}{2\pi} \left(z - \frac{9(1-\lambda)}{2\pi} \sin \left(2\pi \frac{z + \frac{3-5\lambda}{3(1-\lambda)}}{3\lambda} \right) + \frac{4\lambda}{\pi} \right) \\
 &\quad + f'(1) + (f'(0) - f'(1)) \left(\frac{3}{4} - \frac{1}{\pi} \right) + \lambda \\
 1 - \frac{1-\lambda}{4} \leq z \leq 1: f'(z) &= -C_b \frac{1-\lambda}{2\pi} \left(z - \frac{1-\lambda}{2\pi} \sin \left(2\pi \frac{z + \lambda}{1-\lambda} \right) - 1 \right) \\
 &\quad + 1 + f'(1) + (z-1)
 \end{aligned}$$

Fig. 3-34: Modified sine line

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Selectable	Selectable
2. Order (acceleration)	0	0
3. Order (jerk)	0	0

Tab. 3-27: Marginal values for the motion law - Modified sine curve

Extended motion laws

The extended motion laws are special motion laws. They differ from the known motion laws of the VDI guidelines by their new degrees of freedom. While a standard motion law is unambiguously defined by the specification of its left and right marginal values, the motion laws presented allow further different settings, e.g. a required maximum velocity. In this case, the master axis range of the motion step does not have to be indicated. Instead, this results from the parameterization of the motion law and the specification of characteristic values. Thus, for example, acceleration-limited motion allows the distance, starting and end velocity to be specified, but not the setting of a master axis range. The latter, and thus also the duration of the motion step, results from the maximum acceleration and maximum velocity values, which can also be specified.

	Distance	Master axis section	$v_{\max}$	$a_{\max}$	$j_{\max}$
Overshooting-free Polynomial 5th order	Selectable	Selectable	Selectable	Automatically	Automatically
Velocity-limited Polynomial 5th order	Selectable	Selectable	Selectable	Automatically	Automatically
Acceleration-limited Motion	Selectable	Automatically	Selectable	Selectable	Automatically
Velocity-limited Sinoide profile	Selectable	Automatically	Selectable	Selectable	Automatically
Jerk-limited motion	Selectable	Automatically	Selectable	Selectable	Selectable

Tab. 3-28: Extended motion laws

The profile for the **"Velocity-limited polynomial 5th order"** can be as follows:

The polynomial 5th order known from the VDI does not allow for a control of the maximum travel velocity. The motion law is subdivided into three sections to solve this problem. The middle section corresponds to a straight line with the required velocity. This is surrounded by two polynomials 5th order. These ensure that there is a smooth transition from the starting and end velocity to the travel velocity. The intersecting points between the first and second and the second and third sections have to be determined. The points are selected in such a way that no turning point is set for the two polynomials

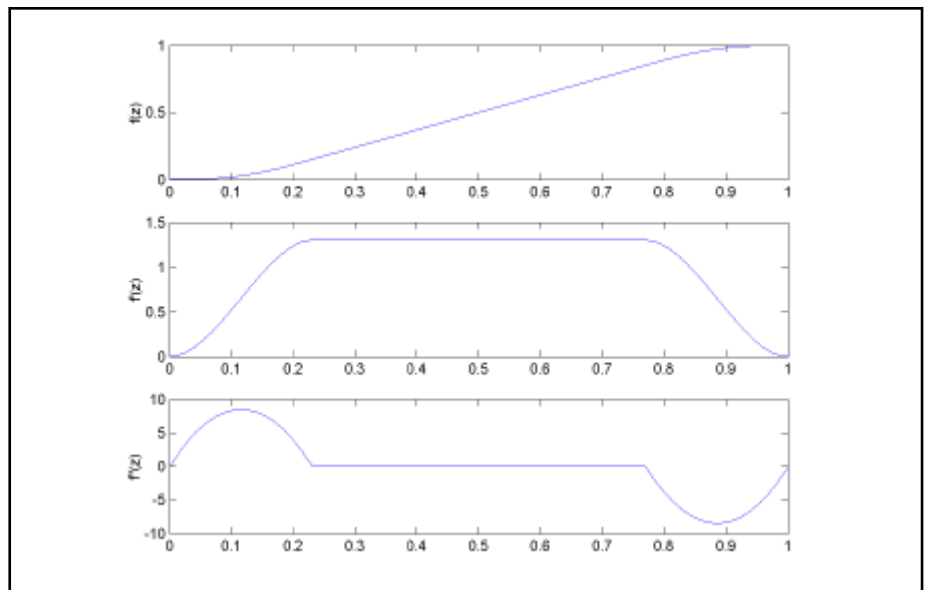


Fig. 3-35: Velocity-limited polynomial 5th order

The profile for **"No overshoot 5th order polynomial"** can be described as follows:

The motion law shown detects and prevents the possible overshoot of the polynomial. The kind of modification depends on the combination of the standardized boundary velocity values. Four cases result. For combinations in which the standardized starting velocity " v_0 " and the standardized end velocity " v_1 " have different signs, this motion law cannot be used. This means that the motion path has a bend at a limit point if it is not able to leave the standardized range of $0 \leq z \leq 1$ and $0 \leq f(z) \leq 1$.

- Case 1:
 $V_0 \leq 1$ and $V_1 \leq 1$

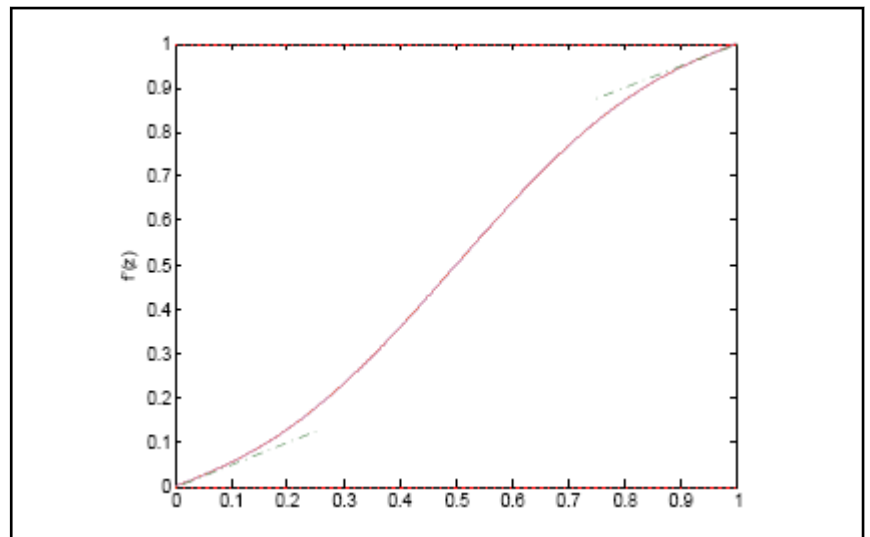


Fig. 3-36: Case 1

- Case 2:
 $V_0 \leq 1$ and $V_1 > 1$

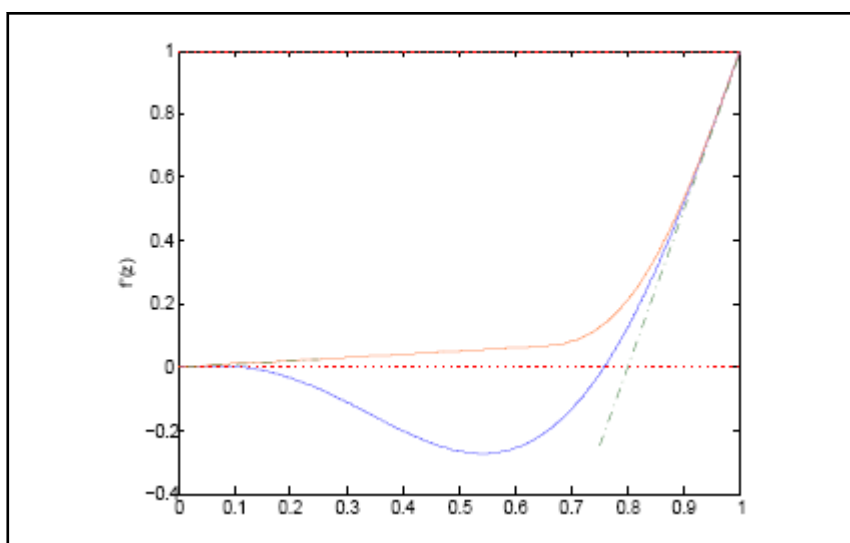


Fig. 3-37: Case 2

- Case 3:
 $V_0 > 1$ and $V_1 \leq 1$

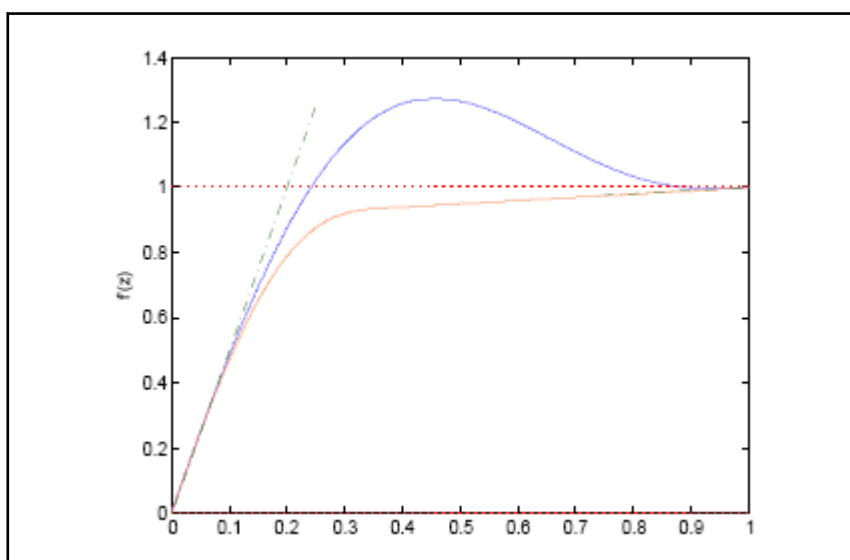


Fig. 3-38: Case 3

- Case 4:
 $V_0 > 1$ and $V_1 > 1$

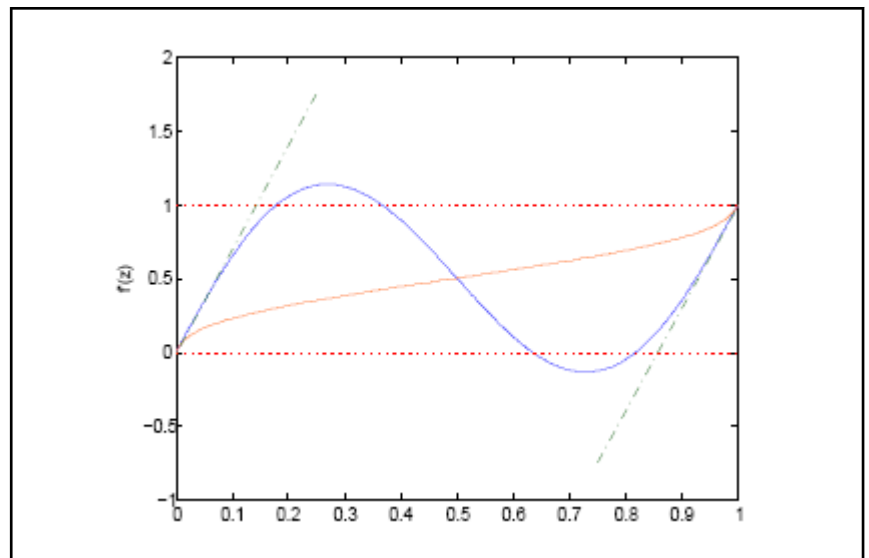


Fig. 3-39: Case 4

The profile for the **"Acceleration-limited motion"** can be described as follows:

The trapezoid profile, often also known as a ramp profile, can be seen as the simplest form of path interpolation. If a way "s" should be traveled, the course of the path can be divided into three phases. In a first phase, in which it is accelerated to a predetermined travel velocity with a predetermined acceleration value, a second phase of constant velocity and finally a deceleration phase in which it is decelerated with a specified delay. The function of the path interpolation is to calculate the starting and end points in time t_1 , t_2 and t_3 and the individual travel times T_1 , T_2 and T_3 from these specified acceleration values a_1 and a_3 as well as the desired velocity v_m and the distance to be traveled s . Furthermore, a required starting velocity v_0 and an end velocity v_3 are considered in the variant presented.

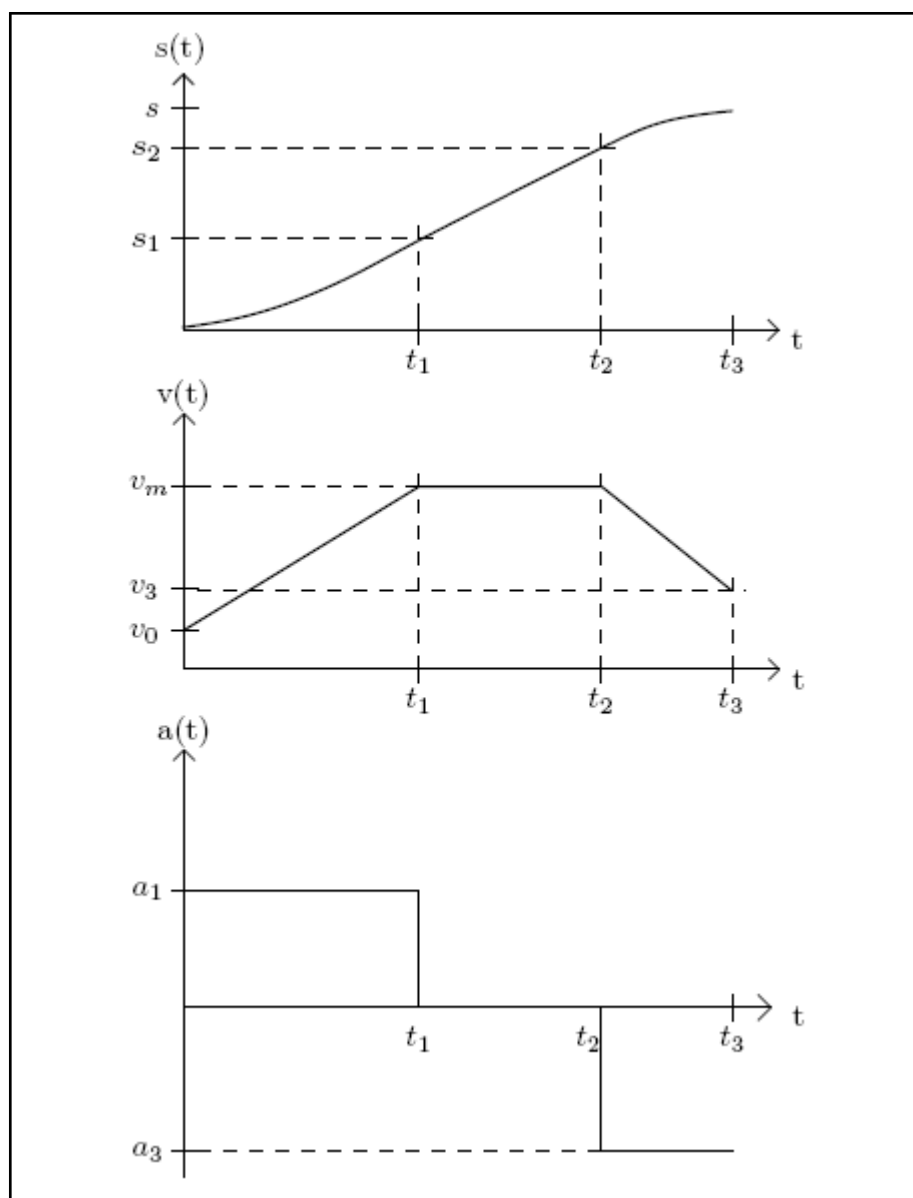


Fig. 3-40: Acceleration-limited motion

The possible marginal values for the motion law are:

Marginal value	z=0	z=1
1. Order (velocity)	Selectable	Selectable
2. Order (acceleration)	Automatically	Automatically
3. Order (jerk)	Automatically	Automatically

Tab. 3-29: Marginal values for the motion law - Acceleration-limited motion

The profile for the **"Acceleration-limited motion with a sinoide profile"** can be described as follows:

With the sinoide profile, the starting and end velocities can be unequal to 0 and have different acceleration factors a_1 and a_3 for the acceleration and deceleration phase. Furthermore, special cases in which the starting velocity v_0 is already higher than the desired travel velocity v_m are considered. The same applies for the end velocity v_3 .

The possible marginal values for the motion law are:

Marginal value	$z=0$	$z=1$
1. Order (velocity)	Selectable	Selectable
2. Order (acceleration)	Automatically	Automatically
3. Order (jerk)	Automatically	Automatically

Tab. 3-30: Marginal values for the motion law - Acceleration-limited motion with sine profile

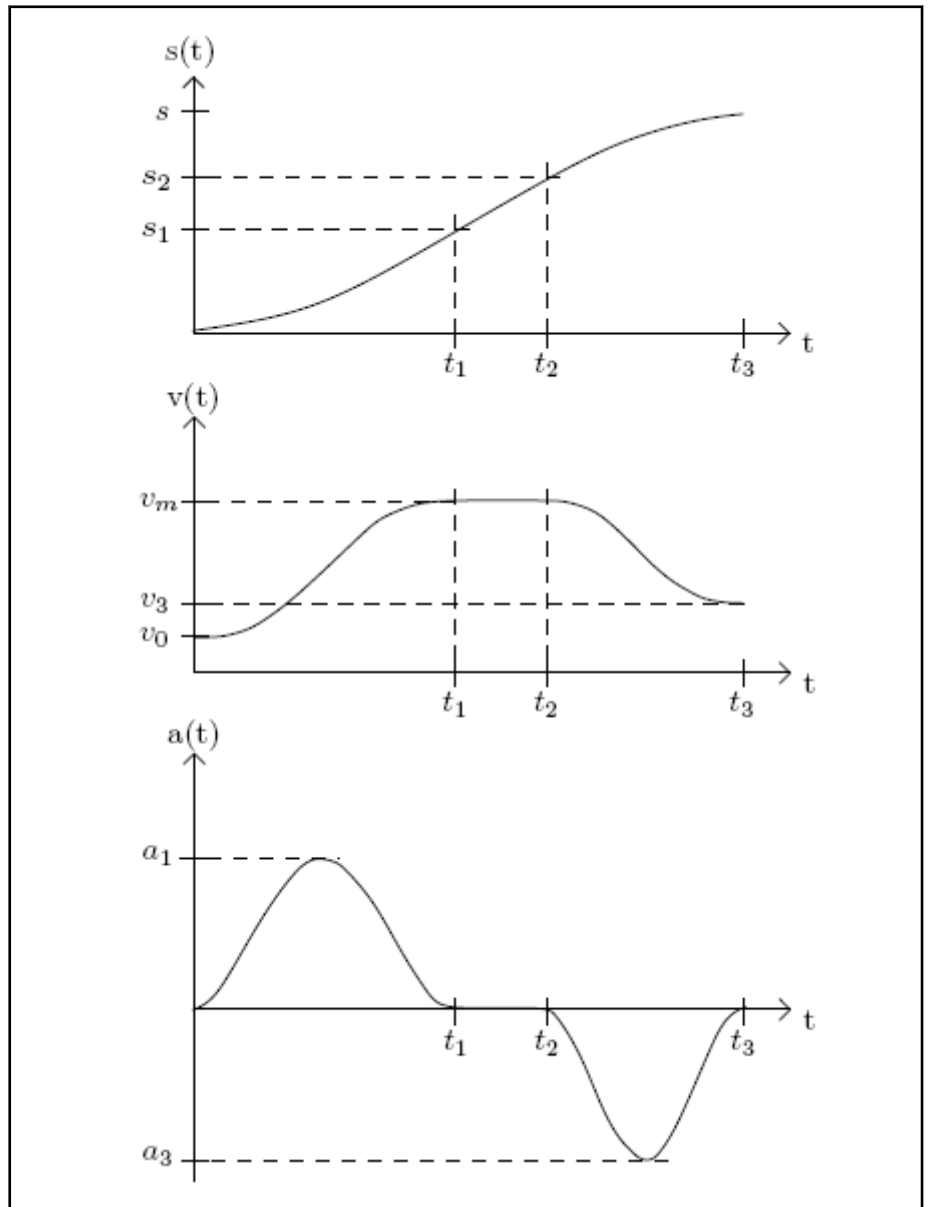


Fig. 3-41: Acceleration-limited motion with sine profile

The profile for the "**Jerk-limited motion**" can be described as follows:

Maintaining acceleration with jerk limitation is applicable to many demanding motion tasks. This applies, for example, to the transportation of fluids and to machines easily susceptible to vibration.

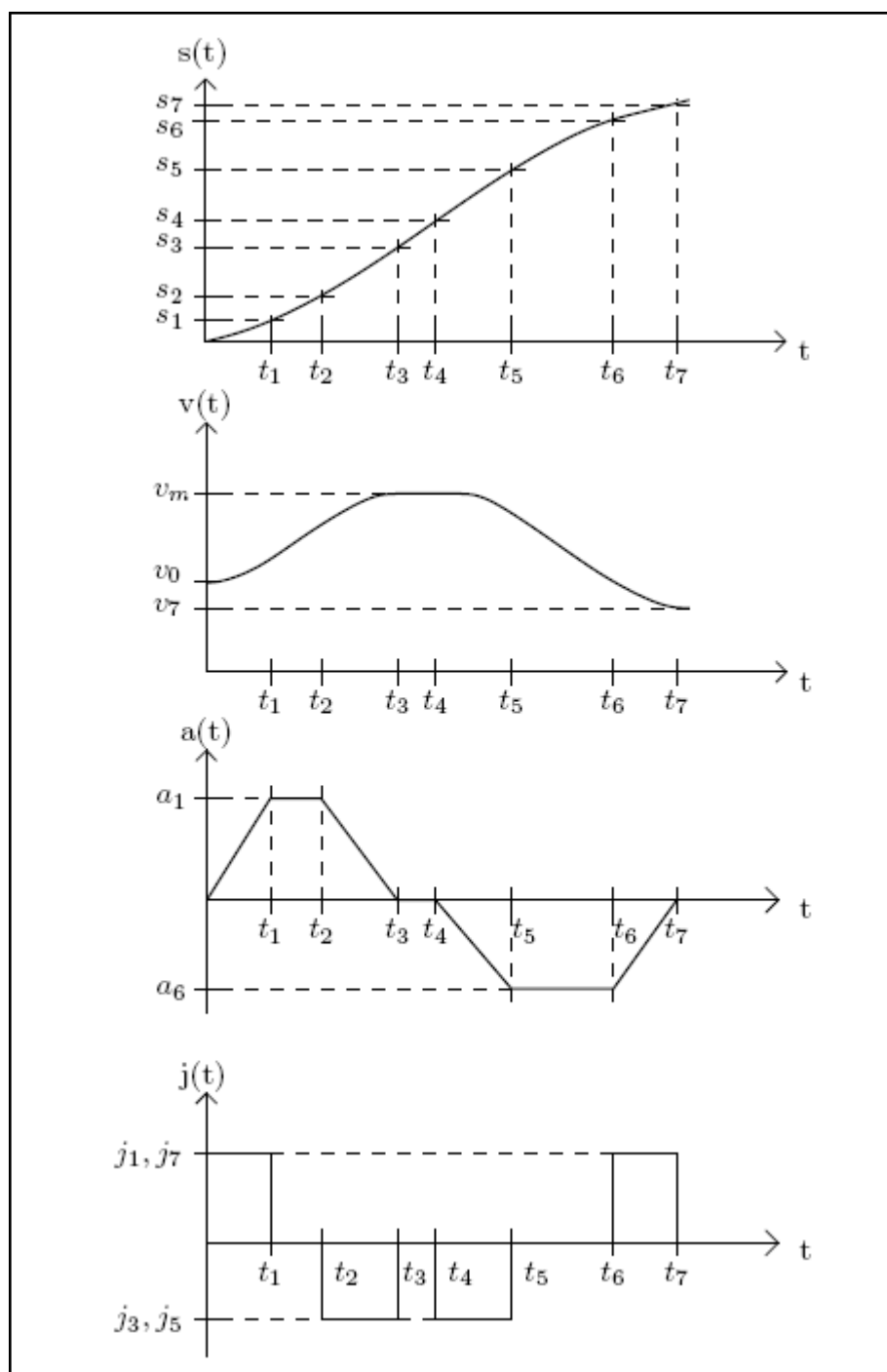


Fig. 3-42: Jerk-limited motion

3.5 Data types to synchronize in the position-controlled synchronous operation modes

3.5.1 Synchronization, introduction and overview

The functions described below are effective in the Phase Synchronization and Electronic Cam operation modes. In general, the slave axis is synchronized after the synchronous operation mode has been activated. This synchronization comprises two steps:

1. First of all, the slave drive accelerates or decelerates to the synchronous velocity with the "synchronization acceleration".
2. In the second step, the position difference is decreased. Therefore, a "synchronization velocity" is superimposed to the synchronous velocity.

Regarding the position adjustment in the second step, it is distinguished between relative and absolute synchronization ([MC_START_MODE](#), [page 51](#)):

- In case of a relative synchronization, the synchronization is completed once the synchronous velocity is reached.
- The absolute angle reference is only determined with the absolute synchronization once the synchronous velocity is reached (synchronization direction, [MC_SYNC_DIRECTION](#), [page 53](#)).

3.5.2 MC_START_MODE

Brief description This parameter defines the type of automatic dynamic synchronization in case of phase synchronization.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of PLCopen

Tab. 3-31: *MC_START_MODE* reference table

Type definition

```

TYPE MC_START_MODE :
(
  ABSOLUTE      := 0, (* synchronization with absolute angle reference, *)
                  (* only for first design                               *)
  RELATIVE      := 1, (* synchronization to synchronous velocity,      *)
                  (* only for first design                               *)
  ABSOLUTE_RAMPIN := 2, (* synchronization with absolute angle reference, *)
                  (* for every design                                *)
  RELATIVE_RAMPIN := 3 (* synchronization to synchronous velocity,      *)
                  (* for every design                                *)
);
END_TYPE

```



For the MB_GearInPos command, only the start modes ABSOLUTE and RELATIVE apply.

Absolute synchronization

The slave axis position in the zero-crossing of the master axis position is unambiguously defined and does not change with every call of MB_MotionProfile/ MC_CamIn.

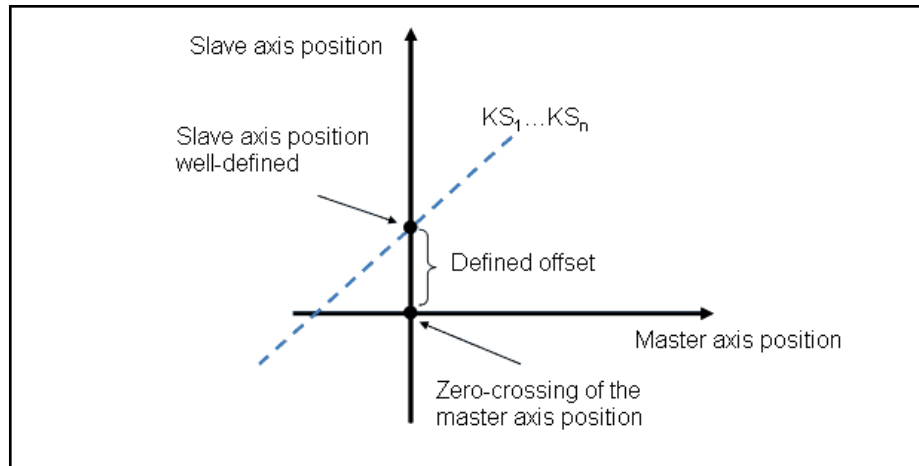


Fig. 3-43: Absolute synchronization

Relative synchronization

The slave axis position in the zero-crossing of the master axis position is arbitrary and changes with every call of MB_MotionProfile/ MC_CamIn.

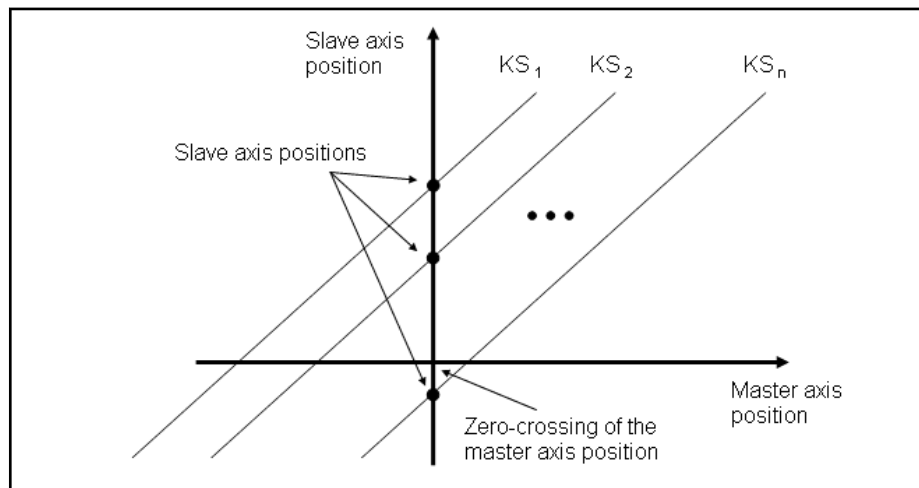


Fig. 3-44: Relative synchronization

"Hard" Synchronization

- StartMode= 0: Absolute synchronization, "hard changeover" in case of a change of profile,
- StartMode= 1: Relative synchronization, "hard changeover" in case of a change of profile

Switching after crossing the master axis switching position (A-0-2900, MotionProfile, master axis switching position) by the master axis.

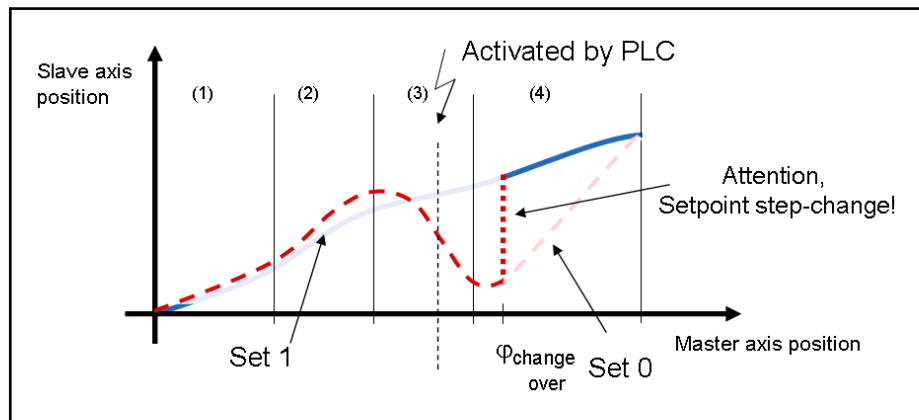


Fig. 3-45: Hard changeover

- "Soft" Synchronization**
- StartMode= 2: Absolute synchronization, "soft changeover" in case of a change of profile,
 - StartMode= 3: Relative synchronization, "soft changeover" in case of a change of profile.

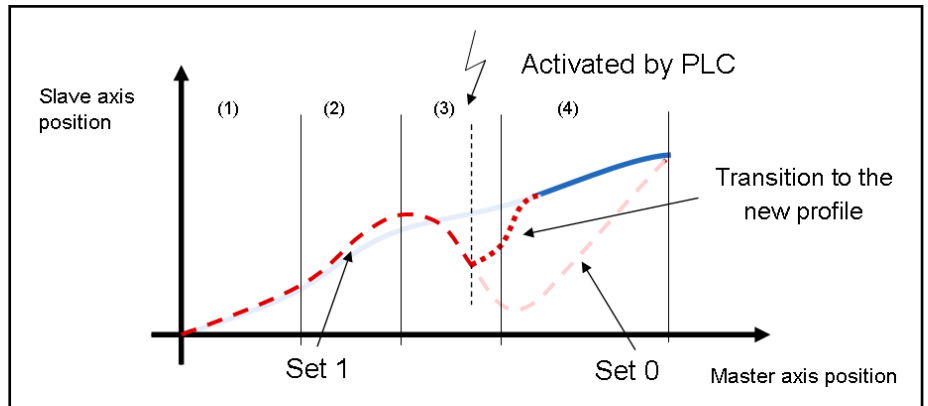


Fig. 3-46: Soft changeover

It is switched immediately after starting the function block and not waited for crossing the master axis switching position (A-0-2900, MotionProfile, master axis switching position) by the master axis.



The "soft synchronization" requires the calculation on the control.

3.5.3 MC_SYNC_DIRECTION

Brief description For modulo axes, the motion of the axis can be in positive or negative direction.

The direction of synchronization defines the direction of the axis motion.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of PLCopen

Tab. 3-32: MC_SYNC_DIRECTION reference table

Type definition

```

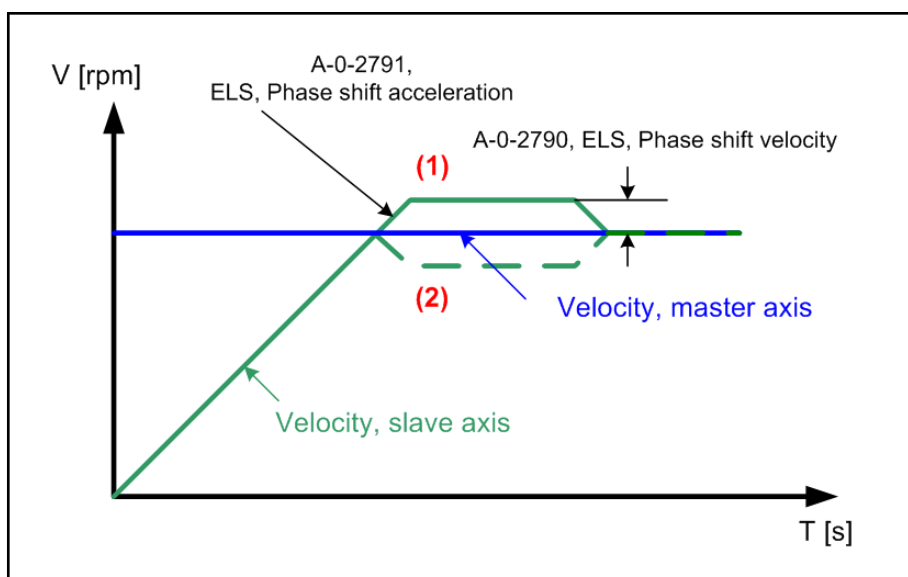
TYPE MC_SYNC_DIRECTION :
(
  SHORTEST_WAY    := 0,  (* kürzester Weg /shortest distance *)
  CATCH_UP        := 1,  (* immer vorwärts / always forward *)
  SLOW_DOWN       := 2
);
END_TYPE

```



If the shortest distance to the absolute synchronization is less than the "Synchronization window" for modulo format, the axis travels the shortest distance and ignores the specified synchronization direction.

Example MLC:



- ① The slave axis follows the master axis
 ② The master axis follows the slave axis

Fig. 3-47: Transition from velocity-synchronous to position-synchronous

In the figure, the slave axis first accelerates to the same velocity as the master axis. Subsequently, the angle difference between the master and the slave axis has to be reduced depending on which one runs in front of the other.

During synchronization, both parameters "A-0-2790, ELS, synchronization velocity" and "A-0-2791, ELS, synchronization acceleration" are applied.

The process is completed when both run velocity-synchronous **and** position-synchronous.

3.6 Data types to diagnose errors in the Motion function blocks

3.6.1 ERROR_CODE

Brief description This enumeration contains all error types characterizing an error at the function block.

The error code provides quick information on the cause of the error.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of POU diagnostics

Tab. 3-33: ERROR_CODE reference table

Type definition

```
(* Enumeration of valid types of diagnosis/ Aufzählung gültiger Fehlerarten *)
TYPE ERROR_CODE :
(
  NONE_ERROR           := 16#0000, (* No error code available *)
  INPUT_INVALID_ERROR  := 16#0001, (* kein Fehlercode verfügbar, FB fehlerfrei *)
  COMMUNICATION_ERROR  := 16#0002, (* Invalid input *)
  RESOURCE_ERROR       := 16#0003, (* Fehler auf Grund ungültiger Eingabewerte *)
  ACCESS_ERROR         := 16#0004, (* Error while communication *)
                        (* Fehler beim Aufbau einer Kommunikation *)
                        (* bzw.während der Kommunikation *)
                        (* Resource error *)
                        (* Ressource (z.B.Parameter) ist nicht vorhanden *)
                        (* Access error *)
)
```

```

(* Ressource ist zwar da, aber man kann nicht *)
(* auf sie zugreifen. *)
(* Beispiel: Man möchte Parameter in Phase 4 be- *)
(* schreiben, der nur in Phase 2 beschreibbar ist*)
STATE_MACHINE_ERROR := 16#0005, (* Invalid state State - Machine *)
(* Der aktuelle Zustand der State-Machine *)
(* verhindert die weitere Abarbeitung *)
INPUT_RANGE_ERROR := 16#0006, (* Error because Input out of range *)
(* Während der Verarbeitung wird festgestellt, *)
(* dass durch eine oder mehrere Eingangsgrößen *)
(* undefinierte bzw. zu große oder zu kleine *)
(* Sollwerte generiert werden. *)
CALCULATION_ERROR := 16#0007, (* Error while calculation *)
(* Ein Berechnungsfehler ist aufgetreten, *)
(* z.B. Division durch 0 *)
DEVICE_ERROR := 16#0008, (* Device Error occurred *)
(* Übergeordneter Gerätefehler ist aufgetreten, *)
(* z.B IndraDrive: *)
(* Kurvenscheibenbetriebsart F2005, *)
(* Ungültige Kurvenscheibe *)
OTHER_ERROR := 16#7FFE, (* Undefined Error *)
(* allgemeiner (undefinierter) Fehler *)
SYSTEM_ERROR := 16#7FFF (* System error occurred *)
(* z.B. Betriebssystemfehler *)
);
END_TYPE

```

3.6.2 ERROR_STRUCT

Brief description The error structure provides any detailed information on an actual error.

This information can be evaluated and read directly at the corresponding instance of a PLCopen function block. The components must be interpreted as follows:

1. Determining table number([ERROR_TABLE](#), page 56). This identifies the origin of the error.
2. Use this finding to evaluate "Additional1". The content is the diagnostic number which can be found in the particular block descriptions (also refer to Troubleshooting Guide, MLC). Check whether "Additional2" = 0; if yes, diagnostic is completed; if no, proceed.
3. Evaluation of "Additional2" (not yet used for MLC_Table).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of POU diagnostics, includes

Tab. 3-34: *ERROR_STRUCT* reference table

Type definition

```

(* Structure for description of diagnosis of function blocks. *)
(* The ERROR_STRUCT is a structure that contains information on the *)
(* corresponding diagnose, which is market by the PLCopen output 'Error' *)
TYPE ERROR_STRUCT :
STRUCT
  Table          : ERROR_TABLE; (* Additional diagnosis table *)
(* links to list used for identification *)
(* of the error of "Additional1" *)
  Additional1    : DWORD; (* Additional diagnosis number1 *)
(* Contains error number according to the *)
(* specified table. *)
  Additional2    : DWORD; (* Additional diagnosis number2 *)
(* Can contain additions to "Additional1". *)
END_STRUCT
END_TYPE

```



In **Additional1**, the messages, warnings and error messages described in "MLC, Diagnostics - Troubleshooting Guide" are displayed.

3.6.3 ERROR_TABLE

Brief description The ERROR_TABLE provides the first element of the [ERROR_STRUCT](#), [page 55](#). Using the number, select the table to be searched for the "Additional1" error code.

Example:

This should for example be "MLC_TABLE", 16#0030 for MLC applications.

The error numbers under "Additional1" contained in function block description get their information from this table.

Function blocks can use the table SERCOS_TABLE, 16#0010, if errors occur during parameter access.

Function blocks can use the tables F_RELATED_TABLE, 16#0170 or INDRV_TABLE, 16#0070 if errors occur when processing the command.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of POU diagnostics, includes

Tab. 3-35: ERROR_TABLE reference table

Type definition

```
(* Enumeration of valid tables in additional information *)
TYPE ERROR_TABLE:
(
  NO_TABLE_USED      := 16#0000, (* reserved *)
  SERCOS_TABLE       := 16#0010, (* Sercos Error: Addtitionall = Sercos Code *)
  MLD_TABLE          := 16#0020, (* Drive-based Motion - Logic *)
  MLC_TABLE          := 16#0030, (* Controller - based Motion - Logic *)
  MTX_TABLE          := 16#0040, (* CNC *)
  MLP_TABLE          := 16#0050, (* PC - based Motion Logic *)
  PLC_TABLE          := 16#0060, (* PLC *)
  INDRV_TABLE        := 16#0070, (* IndraDrive *)
  DIAX_TABLE         := 16#0080, (* DIAX *)
  ECO_TABLE          := 16#0090, (* EcoDrive *)
  PB_DP_TABLE        := 16#0130, (* Profibus *)
  DEVICENET_TABLE    := 16#0140, (* DeviceNet *)
  ETHERNET_TABLE     := 16#0150, (* EtherNet *)
  ETHERNET_IP_TABLE  := 16#0151, (* EtherNet/IP ODVA *)
  INTERBUS_TABLE     := 16#0160, (* Interbus *)
  F_RELATED_TABLE    := 16#0170, (* function related *)
  CAN_TABLE          := 16#0180, (* CAN bus *)
  INLINEIO_TABLE     := 16#0190, (* Inline IOs *)
  PROFINETIO_TABLE   := 16#0200, (* ProfinetIO, ProfinetIO Device *)

  USER1_TABLE        := 16#1000, (* free User Table *)
  ...
  USER10_TABLE       := 16#1009 (* free User Table *)
);
END_TYPE
```


3.7 Data types to report date and absolute time specifications for Motion function blocks

3.7.1 Reporting date and absolute time specification, introduction and overview

For the indication of the date and other absolute time designations, there are three structures:

- [SYS_TIME64, page 57](#), system time in microseconds since 01.01.1970, 00:00 o'clock.
- [EXT_SYS_TIME_DATE, page 57](#), system time in microseconds since 01.01.1970, 00:00 o'clock, additional data year, month, day, hours, minutes, seconds, milliseconds and week day are calculated from the system time.
- [EXT_SYS_TIME_DATE, page 57](#), month, day, hours, minutes, seconds, milliseconds and week day, calculated on the basis of the system time in microseconds since 01.01.1970, 00:00 o'clock.

3.7.2 SYS_TIME64

Brief description Structure for the indication of the system time in microseconds since 01.01.1970, 00:00 o'clock.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of times

Tab. 3-36: *SYS_TIME64 reference table*

Type definition

```
(* Structure of system time - Microseconds since 1970 *)
TYPE SYS_TIME64 :
STRUCT
  LowValue:    DWORD;    (* Low value [us]      *)
  HighValue:   DWORD;    (* High value [us]   *)
END_STRUCT
END_TYPE
```

3.7.3 EXT_SYS_TIME_DATE

Brief description Structure to return the system time in microseconds since 01.01.1970, 00:00 o'clock, additional data year, month, day, hours, minutes, seconds, milliseconds and week day are calculated from the system time.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/data types of times

Tab. 3-37: *EXT_SYS_TIME_DATE reference table*

Type definition

```
(* Extended structure of system time *)
TYPE EXT_SYS_TIME_DATE :
STRUCT
  LowValue: DWORD;    (* Low value [us]      *)
  HighValue: DWORD;   (* High value [us]   *)
END_STRUCT
```

```

Year : UINT;          (* Year          *)
Month : UINT;         (* Month         *)
Day : UINT;           (* Day           *)
Hour : UINT;          (* Hour          *)
Minute : UINT;        (* Minute        *)
Second : UINT;        (* Second        *)
Milliseconds : UINT;  (* Milliseconds  *)
DayOfWeek: UINT;      (* Day of week   *)
END_STRUCT
END_TYPE

```

3.7.4 SYS_TIME_DATE

Brief description Structure for the indication of the information year, month, day, hours, minutes, seconds, milliseconds and week day which are calculated from a system time in microseconds since 01.01.1970, 00:00 o'clock.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/data types of times
PLC basic package	

Tab. 3-38: SYS_TIME_DATE reference

Type definition

```

(* Structure of system time *)
TYPE SYS_TIME_DATE :
STRUCT
  Year : UINT;          (* Year          *)
  Month : UINT;         (* Month         *)
  Day : UINT;           (* Day           *)
  Hour : UINT;          (* Hour          *)
  Minute : UINT;        (* Minute        *)
  Second : UINT;        (* Second        *)
  Milliseconds : UINT;  (* Milliseconds  *)
  DayOfWeek: UINT;      (* Day of week   *)
END_STRUCT
END_TYPE

```

3.8 Data types to support touch probe functions

3.8.1 Data types to support touch probe functions, general information

The section contains data types supporting the touch probe functions during the processing in the drive and in the control as well:

- [MB_DIFF_EVAL](#), page 59, setting of the methods for differential measuring
- [MB_EXPECT_WINDOW_MODE](#), page 60, configuration and activation of the expectation window
- [MB_INTERPOLATION_MODE](#), page 60, activation of the interpolation
- [MB_POSITION_EVAL](#), page 61, selection and activation of the trigger signal
- [MB_PROBE_MODULE](#), page 61, selection of the trigger source for the touch probe
- [MB_PROBE_INPUT](#), page 61, selection of the desired input on the selected module
- [MB_PROBE_DATA](#), page 62, selection of the reference signal source of the touch probe

The data belonging to a touch probe function is summarized in the following structure:

- [MB_PROBE_CONFIG](#), page 63

3.8.2 MB_DIFF_EVAL

Brief description The enumeration type "MB_DIFF_EVAL" supports the setting of the methods for difference measuring.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/TouchProbe

Tab. 3-39: MB_DIFF_EVAL reference table

Type definition

```

TYPE MB_DIFF_EVAL :
(
  DIFF_DISABLED      := 0, (* diff.measurement is disabled *)
  DIFF_STANDARD_MODE := 1, (* diff. last two measured values*)
  DIFF_NEG_TO_NEG_EDGE:= 2, (* neg. edge to neg. edge diff. *)
  DIFF_POS_TO_NEG_EDGE:= 3, (* pos. edge to neg. edge diff. *)
  DIFF_NEG_TO_POS_EDGE:= 4, (* neg. edge to pos. edge diff. *)
  DIFF_POS_TO_POS_EDGE:= 5 (* pos. edge to pos. edge diff. *)
);
END_TYPE

```

Functional description The following settings are supported:

- 0 - Difference measurement is switched off, difference signals for ML_TouchProbe and/or ML_TouchProbe_Continuous are not calculated.
- 1 - Difference between the last two measured values, depending on the activated edge.
Example: Both edges are activated, the difference pos/neg edge, then neg/pos edge, pos/neg edge is registered.
- In the following figure, the extended methods 2..5 of the difference measurement are displayed:

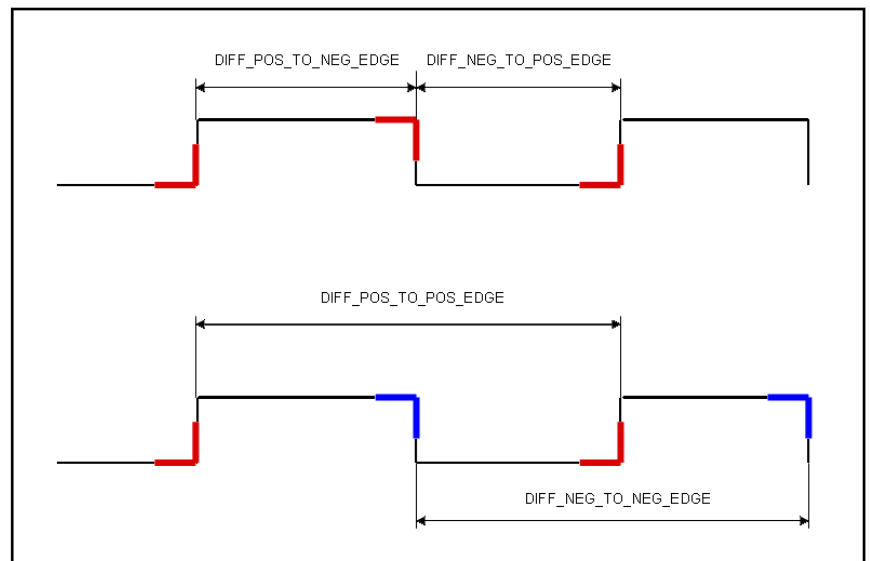


Fig. 3-48: Difference measurement methods

3.8.3 MB_EXPECT_WINDOW_MODE

Brief description The enumeration type "MB_EXPECT_WINDOW_MODE" supports the configuration and activation of the expectation window.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/TouchProbe

Tab. 3-40: MB_EXPECT_WINDOW_MODE reference table

Type definition

```

TYPE MB_EXPECT_WINDOW_MODE :
(
  EXPECT_WINDOW_DISABLED:= 0, (* expectation window disabled *)
  EXPECT_WINDOW_ABS      := 1, (* expectation window absolute *)
  EXPECT_WINDOW_REL      := 2, (* expectation window relative *)
                                (* to "ExpectWindowSetPoint" *)
);
END_TYPE

```

Functional description The user can decide separately for the positive and the negative edge of the touch probe value for an expectation window.

If the user changes the decision for the use of an expectation window, he can define the limits thereof in two different ways:

1. Expectation window disabled:
The respective edge of the touch probe signal is used unfiltered.
2. Expectation window defined absolute:
The start position and the end position of the expectation window have to be indicated as absolute values. The center point value is meaningless in this case.
3. Expectation window defined relative:
The center point value of the expectation window has to be indicated as an absolute value. Starting and end position indicate the distance to the center point value.

3.8.4 MB_INTERPOLATION_MODE

Brief description The enumeration type "MB_INTERPOLATION_MODE" supports the activation of the interpolation.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/TouchProbe

Tab. 3-41: MB_INTERPOLATION_MODE reference table

Type definition

```

TYPE MB_INTERPOLATION_MODE:
(
  NO_INTERPOLATION      := 0, (* no interpolation *)
  LINEAR_INTERPOLATION  := 1, (* linear interpolation *)
  CUBIC_3_INTERPOLATION := 2, (* cubic interpolation with 3 setpoints *)
  CUBIC_4_INTERPOLATION := 3, (* cubic interpolation with 4 setpoints *)
);

```

```
);
END_TYPE
```

3.8.5 MB_POSITION_EVAL

Brief description The enumeration type "MB_POSITION_EVAL" supports the selection and activation of the trigger signal.

Use these enumeration values to select whether the positive, negative or both edges are triggered.



The default value "Zero" blocks triggering!

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/TouchProbe
PLC basic package	

Tab. 3-42: MB_POSITION_EVAL reference table

Type definition

```
TYPE MB_POSITION_EVAL :
(
  POSITION_EVAL_DISABLED:= 0, (* disabled *)
  POSITION_POS_EDGE      := 1, (* positive edge is evaluated *)
  POSITION_NEG_EDGE      := 2, (* negative edge is evaluated *)
  POSITION_POS_NEG_EDGE := 3 (* pos.and neg. edge are evaluated *)
);
END_TYPE
```

3.8.6 MB_PROBE_MODULE

Brief description The enumeration type "MB_PROBE_MODULE" supports the selection of the trigger source for the touch probe.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/TouchProbe
PLC basic package	

Tab. 3-43: MB_PROBE_MODULE reference table

Type definition

```
TYPE MB_PROBE_MODULE :
(
  ONBOARD_MODULE := 0, (* Onboard module MLC, Inputs 1...8 *)
  FUNCTION_MODULE_1:= 1, (* Function module 1 MLC, Inputs 1..16 (CFL-01.1-E2, Fast IO) *)
  FUNCTION_MODULE_2:= 2, (* Function module 2 MLC, Inputs 1..16 (CFL-01.1-E2, Fast IO) *)
  FUNCTION_MODULE_3:= 3, (* Function module 3 MLC, Inputs 1..16 (CFL-01.1-E2, Fast IO) *)
  FUNCTION_MODULE_4:= 4 (* Function module 4 MLC, Inputs 1..16 (CFL-01.1-E2, Fast IO) *)
);
END_TYPE
```

3.8.7 MB_PROBE_INPUT

Brief description The enumeration type "MB_PROBE_INPUT" supports the selection of the desired input on the selected module.



For the OnBoard module, the inputs 1..8 are available.

The Fast I/O modules are provided with the inputs 1..16 as trigger source.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/TouchProbe
PLC basic package	

Tab. 3-44: MB_PROBE_INPUT reference table

Type definition

```

TYPE MB_PROBE_INPUT :
(
  D_INPUT_1  := 1,  (* digital input 1 *)
  D_INPUT_2  := 2,  (* digital input 2 *)
  D_INPUT_3  := 3,  (* digital input 3 *)
  D_INPUT_4  := 4,  (* digital input 4 *)
  D_INPUT_5  := 5,  (* digital input 5 *)
  D_INPUT_6  := 6,  (* digital input 6 *)
  D_INPUT_7  := 7,  (* digital input 7 *)
  D_INPUT_8  := 8,  (* digital input 8 *)
  D_INPUT_9  := 9,  (* digital input 9 *)
  D_INPUT_10 := 10, (* digital input 10 *)
  D_INPUT_11 := 11, (* digital input 11 *)
  D_INPUT_12 := 12, (* digital input 12 *)
  D_INPUT_13 := 13, (* digital input 13 *)
  D_INPUT_14 := 14, (* digital input 14 *)
  D_INPUT_15 := 15, (* digital input 15 *)
  D_INPUT_16 := 16, (* digital input 16 *)
);
END_TYPE

```

3.8.8 MB_PROBE_DATA

Brief description The enumeration type "MB_PROBE_DATA" supports the selection of the reference signal source of the touch probe for the touch probes of the MLC system (A-parameters) and for the touch probes of the drive (S-/P-parameter).

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS	RIL_CommonTypes/TouchProbe
PLC basic package	

Tab. 3-45: MB_PROBE_DATA reference table

Type definition

```

TYPE MB_PROBE_DATA :
(
  TP_ACT_POS_FEEDBACK1 := 0, // S-0-0051 / A-0-0100, Position, feedback 1 v
  TP_ACT_POS_FEEDBACK2 := 1, // S-0-0053 / -----, Position, feedback 2 v
  TP_ACT_POSIN_ACT_CYCLE := 2, // P-0-0753 / -----, Position actual value
  TP_SYNC_CMD_POS := 3, // P-0-0778 / A-0-2653, Synchronous position c
  TP_PROBE_TIME := 4, // ----- / -----, time measurement
  TP_ACT_MEASURING_ENC_POS := 5, // P-0-0052 / -----, Actual position value
  TP_RES_MASTER_AXIS_POS := 6, // P-0-0775 / A-0-2650, Resulting master axis p
  TP_EFF_MASTER_AXIS_POS := 7, // P-0-0776 / A-0-2651, Effective master axis p
  TP_CAM_SHAFT_PROF_ACCESS_ANGLE := 8, // P-0-0227 / A-0-2656, Cam shaft profile acces
  TP_ACT_MASTER_AXIS_POS := 9, // not yet supported!!!
                                // P-0-0053 / A-0-0300, Master axis, actual po
  TP_ELS_CMD_POS := 10, // ----- / A-0-2655, ELS, position command
  TP_USER_DATA_CONTAINER_A := 11, // ----- / A-0-0521, Container of actual use
  TP_USER_DATA_CONTAINER_B := 12, // ----- / A-0-0523, Container of actual use
  TP_USER_DATA_CONTAINER_C := 13, // ----- / A-0-0525, Container of actual use
  TP_USER_DATA_CONTAINER_D := 14, // ----- / A-0-0527, Container of actual use
  TP_USER_DATA_CONTAINER_E := 15, // ----- / A-0-0529, Container of actual use
  TP_USER_DATA_CONTAINER_F := 16, // ----- / A-0-0531, Container of actual use

```

```
);
END_TYPE
```

3.8.9 MB_PROBE_CONFIG

Brief description The structure "MB_PROBE_CONFIG" is used for registering the data of a touch probe function.

Assignment: Target system/library

Target system	Library/folder
MLC 15VRS PLC basic package	RIL_CommonTypes/TouchProbe

Tab. 3-46: MB_PROBE_CONFIG reference table

Type definition

```
TYPE MB_PROBE_CONFIG :
STRUCT
  ModuloValueDrive      :REAL;      (* Modulo value *)
  ProcessingFormat      :BOOL;      (* Weighting position, abs./modulo *)
  ProbePositionEval     :MB_POSITION_EVAL; (* Enabling trigger signal, de
                                     (* pos., neg., both edges *)
  DifferenzMeasurement  :MB_DIFF_EVAL;  (* difference measurement meth
  ExpectWindowPosMode   :MB_EXPECT_WINDOW_MODE; (* Expectation window mode pos
                                     (* def.: Off, absolute / relat
  ExpectWindowNegMode   :MB_EXPECT_WINDOW_MODE; (* Expectation window mode neg
                                     (* def.: Off, absolute / relat
  ExpectWindowPosStart  :REAL;      (* Start pos. expectation window pos. edge
  ExpectWindowPosEnd    :REAL;      (* End pos. expectation window pos. edge *)
  ExpectWindowNegStart  :REAL;      (* Start pos. expectation window neg. edge
  ExpectWindowNegEnd    :REAL;      (* End pos. expectation window neg. edge *)
END_STRUCT
END_TYPE
```

"MB_PROBE_CONFIG" uses the following data types

- [MB_POSITION_EVAL](#), page 61
- [MB_DIFF_EVAL](#), page 59
- [MB_EXPECT_WINDOW_MODE](#), page 60

3.8.10 MB_PROBE_CONFIG, special features of Sercos-based axes

Functional description: The following table reflects the connection to the parameter system of the control:

Structure element	Data type	Parameters
ModuloValueDrive	REAL	A-0-0045, Modulo value
ProcessingFormat	BOOL	A-0-0059.7, Scaling factor for velocity data, bit 7 absolute/modulo
ProbePositionEval	MB_POSITION_EVAL , page 61	Activation positive/negative edge, M-0-0007, Control word
Differential measuring	MB_DIFF_EVAL , page 59	Differential measuring, M-0-0007, Control word
ExpectWindowPosMode	MB_EXPECT_WINDOW_MODE , page 60	Activation and configuration of the expectation window of the positive edge M-0-0007, Control word
ExpectWindowNegMode	MB_EXPECT_WINDOW_MODE , page 60	Activation and configuration of the expectation window of the negative edge M-0-0007, Control word

Structure element	Data type	Parameters
ExpectWindowPosStart	REAL	M-0-0202, Starting position of the expectation window, positive edge
ExpectWindowPosEnd	REAL	M-0-0203, End position of the expectation window, positive edge
ExpectWindowNegStart	REAL	M-0-0204, Starting position of the expectation window, negative edge
ExpectWindowNegEnd	REAL	M-0-0205, End position of the expectation window, negative edge

Tab. 3-47: Parameter assignment touch probe function

4 RIL_Utilities.library

4.1 Introduction and overview

General information The "RIL_Utilities" library extends the basic functionality of the MLC system and provides common basic functionalities. The library includes basic functionalities that can be used for several applications or as standalone functionality.



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

Areas of application and functions/function blocks of RIL_Utilities

Area of application (folder in library)	
Function blocks	Description
Jogging variables (adjustment)	
IL_ContinuousAdjustType01, page 72	Continuous adjustment of a REAL variable via binary inputs
IL_ContinuousAdjustType02, page 75	Continuous adjustment of a DINT variable via binary inputs.
IL_IncrementalAdjustType01, page 78	Incremental adjustment of a REAL variable via binary inputs
Basic classes for the function block development (Base FB)	
Basic function block IL_EnableDoneBase, page 86	Implements the interface of a function block with an "Enable" input and "Done", "Active" and "Standard" error outputs
Basic function block IL_EnableInOpBase, page 89	Implements the interface of a function block with an "Enable" input and "InOperation" and "Standard" error outputs
Basic function block IL_EnableInOpShutdown-Base, page 91	Implements the interface of a function block with an "Enable" input and "InOperation", "Shutdown" and "Standard" error outputs.
Basic function block IL_ExecuteDoneAborted-Base, page 100	Implements the interface of a function block with an "Execute" input and the "Done", "CommandAborted", "Active" and "Standard" error outputs
Basis function block IL_ExecuteDoneAborted-ShutdownBase, page 102	Implements the interface of a function block with an "Execute" input and "Done", "Active", "Shutdown", "CommandAborted" output and "Standard" error outputs
Basic function block IL_ExecuteDoneBase, page 94	Implements the interface of a function block with an "Execute" input and "Done", "Active" and "Standard" error outputs
Basic function block IL_ExecutelnOpBase, page 97	Implements the interface of a function block with an "Execute" input and "InOperation", "Active" and "Standard" error outputs
Conversion (bit-stream "connect")	
IL_BoolToByte, page 108	8 individual bits (BOOL) are converted to one BYTE (FUN)
IL_BoolToDWord, page 109	32 individual bits (BOOL) are converted to one DWORD (FUN)
IL_BoolToWord, page 111	16 individual bits (BOOL) are converted to one WORD (FUN)
IL_ByteToDWord, page 112	4 BYTES are converted to one DWORD (FUN)
IL_ConcatByte, page 113	2 BYTES are summarized to one WORD (FUN)
IL_ConcatWord, page 114	2 WORDs are summarized to one DWORD
Conversion (bit stream "split")	

IL_ByteToBool, page 115	Function block extracts 8 bits (BOOL) from one BYTE
IL_DWordToBool, page 116	Function block extracts 32 bits (BOOL) from one DWORD
IL_DWordToByte, page 119	Function block extracts 4 BYTEs from one DWORD
IL_HighByte, page 119	FUN extracts a high-order BYTE from one WORD
IL_HighWord, page 120	FUN extracts a high-order WORD from one DWORD
IL_LowByte, page 120	FUN extracts a low-order BYTE from one WORD
IL_LowWord, page 121	FUN extracts a low-order WORD from one DWORD
IL_WordToBool, page 121	Function block extracts 16 bits (BOOL) from one WORD
Conversion (ByteSwap)	
IL_SwapDWord, page 123	FUN exchanges BYTEs of a DWORDs (3 2 1 0 -> 0 1 2 3) Note: Not PLC-compatible!
IL_SwapWord, page 124	FUN exchanges BYTEs of a WORD (1 0 -> 0 1)
Checksums (CRC16)	
IL_CRC16, page 125	Function block computes CRC16 checksum using a date
Conversion (data type "change")	
IL_AsciiToByte on page 126	FUN converts any ASCII character to a byte code
IL_ByteToAscii on page 126	FUN converts any byte code to an ASCII character
IL_RealToString on page 127	FUN converts REAL value to a STRING
IL_LRealToString on page 127	FUN converts (L)REAL value format-controlled to a STRING
High resolution time support (debugging)	
IL_HighResTimeDiff, page 129	Calculates the time difference of two high-resolution time cycles of the system in microseconds (before and after a code section)
IL_HighResTimeTick, page 130	Reads the high-resolution time cycle of the system
Trigger (FlipFlops)	
IL_Toggle, page 131	Settable and resettable FlipFlop
Mathematical functions	
IL_InterpolationLinear, page 133,	Calculates intermediate points and linearizations for example
IL_MaxValue, page 135,	Indicates the maximum value of all occurred values
Character strings without length limitation (StringFun_NoLimitStrLength)	
IL_StrLower, page 139	Function converts character string to lower case letters
IL_StrUpper, page 140	Function converts character string to upper case letters
IL_StrCompare, page 141	Function compares two character strings and returns the difference. Upper and lower case letters considered
IL_StrCompareNoCase, page 141	Function compares two character strings and returns the difference. Upper and lower case letters not considered
IL_StrNCompare, page 142	Function compares the entered number of characters of two character strings and returns the difference. Upper and lower case letters considered

IL_StrNCompareNoCase, page 143	Function compares the entered number of characters of two character strings and returns the difference. Upper and lower case letters not considered
IL_AsciiStreamToBase64, page 143	Function converts a character string, saved in ASCII format, to the BASE64 format
IL_Base64ToAsciiStream, page 145	Function converts a character string, saved in BASE64 format, to the ASCII format
IL_BufferToString, page 147	Function copies an ASCII character string from a byte array
IL_StringToBuffer, page 148	Function copies an ASCII character string to a byte array
IL_StrCopy, page 150	Function copies character string from the source to the target address
IL_StrNCopy, page 151	Function copies partial character string from a character string to a target character string. Length of the target character string is specified. If the source character string is shorter than the requested length, the target character string is supplemented by fill characters
IL_CONCAT, page 152	Function adds character string 2 behind character string 1
IL_DELETE, page 153	Function deletes a partial character string from the character string of a specified position
IL_FIND, page 154	Function searches for position of entered partial character string in a character string
IL_INSERT, page 154	Function inserts a partial character string to a character string starting from the entered position
IL_LEFT, page 155	Function provides the left part of a character string with a length of x characters
IL_LEN, page 156	Function provides the character string length
IL_MID, page 156	Function provides a partial character string to a character string starting from the entered position
IL_REPLACE, page 157	Function replaces a partial character string of a character string by another partial character string
IL_RIGHT, page 158	Function provides the right part of a character string with a length of x characters
IL_Ascii_To_Char, page 159	Function converts ASCII code to a character
IL_Char_To_Ascii, page 160	Function converts a character to ASCII code
IL_Int_To_Str, page 160	Function converts (depending on format) an integer to a character string
IL_Real_To_Str, page 161	Function converts (depending on format) a REAL number to a character string
IL_String_To_Unicode, page 162	Function converts a character string according to the specified option (utf8/utf16) into Unicode and stores the values in a DWORD array
IL_Unicode_To_String, page 163	Function converts a Unicode character string to a STRING according to the specified option (utf8/utf16)
IL_BufferGetElementCount, page 164	The function specifies the number of objects in the character string separated by a "separator" character.
IL_Code, page 165	Function determines the ASCII code of a specified character in a character string in decimal representation

IL_StrInverse, page 166	Function inverts a character string
IL_StrSetEndOfString, page 166	Functions sets the end of a character string
IL_StrGetElement, page 167	The function is split into multiple elements by the entered separator string. The function calls one of the elements
IL_StrLChr, page 168	Function isolates a partial character string from the character string up to the first occurrence of the entered separator
IL_StrRChr, page 169	Function isolates a partial character string from the character string up to the last occurrence of the entered separator
IL_StrSetElement, page 170	Character string separated into elements by the entered separator order. The function positions a new element to the desired position in the character string
IL_StrSplit, page 171	Character string separated by specified separator. The function stores the address of each partial character string in an array
IL_StrTrimAll, page 172	Function deletes all specified characters in a character string
IL_StrTrimSpace, page 173	Depending on the option, the function deletes spaces in a character string
Time functions (TIME)	
IL_Date, page 175	Reads the current system date
IL_DateAndTime, page 175	Reads current system date and time
IL_ExtSysTimeDate, page 176	Reads current system date and time in the extended system format
IL_LocalTime, page 176	Converts control-internal UTC time to local time
IL_SysTime64, page 178	Reads current system date and time in microseconds since 01.01.1970
IL_SysTime64ToSysTimeDate, page 179	Converts the format of system date and system time
IL_SysTimeDate, page 179	Reads current system date and time
IL_SysTimeDateToSysTime64, page 180	Converts the format of system date and system time
IL_TimeOfDay, page 180	FUN reads the current system time
Double-linked lists (Utilities/Logic)	
IL_LinkedList, page 184,	Manages double-linked lists (used for ring buffer, FIFOs and LIFOs)
Modulo value calculation (Utilities/Logic)	
IL_GetModuloType01, page 190	Calculates a modulo value. Value range between 0.0 and + modulo value (REAL)
IL_GetModuloType02, page 191	Calculates a modulo value. Value range between 0 and + modulo value (DINT)
IL_GetModuloType03, page 192	Calculates a modulo value. Value range between - modulo value and + modulo value (REAL)
IL_GetModuloType04, page 193	Calculates a modulo value. Value range between - modulo/2 and + modulo value/2 (REAL)

Tab. 4-1: RIL_Uilities library

4.2 Jogging variables (adjustment)

4.2.1 Introduction and overview

Adjust (jog) the PLC variables continuously or incrementally via binary inputs using the following function blocks:

Area of application (folder in library)	
Function blocks	Description
Jogging variables (adjustment)	
IL_ContinuousAdjustType01, page 72	Continuous adjustment of a REAL variable via binary inputs
IL_ContinuousAdjustType02, page 75	Continuous adjustment of a DINT variable via binary inputs
IL_IncrementalAdjustType01, page 78	Incremental adjustment of a REAL variable via binary inputs

Tab. 4-2: *RIL_Uilities, Adjustment*



The mentioned function blocks changed their names in 14VRS from MB_* to IL_* and moved from "ML_TechBase" to "RIL_Uilities".

When using the above mentioned function blocks to jog PLC variables, consider the following requirements:

- Stop the operation via a binary input ("Enable")
- The variable to be affected can be continuously adjusted using the function blocks IL_ContinuousAdjustType01 and IL_ContinuousAdjustType02 (similar to the long jog for SYNAX)
- The variable to be affected can be incrementally adjusted using the function block IL_IncrementalAdjustType01 (similar to the short jogging for SYNAX)
- The selected variable can either be changed in incremented or decremented steps within the limit values
- When reaching a limit value "HighLimitActive" = TRUE or "LowLimitActive" = TRUE), the continuous adjustment is disabled and thus, the function block does no longer affect the variable. In this case, the respective limit value is displayed at the "HighLimit" input or at the "LowLimit" input

General basics

If the limits "HighLimit" and "LowLimit" have the same value for a specified modulo value, "LowLimit" is set to zero and "HighLimit" is set to the modulo value. Thus, it is possible to adjust the limit values in the entire modulo range.

The change rate describes the rate the affected variable changes during adjustment. This rate depends on the step width and the number of steps per second. The change rate is calculated as follows, for example, when changing a position:

$$\text{AlterationVelocity} \left[\frac{\text{Degree}}{\text{s}} \right] = \text{Increments} [\text{Degree}] * \text{Increments per Second} \left[\frac{1}{\text{s}} \right]$$

Fig. 4-1: *Change rate when changing a position*

If, for example, the speed of the virtual master axis is changed, the change rate is calculated as follows:

$$\text{AlterationVelocity} \left[\frac{r.p.m.}{s} \right] = \text{Increments} [r.p.m.] * \text{Increments per Second} \left[\frac{1}{s} \right]$$

Fig. 4-2: Change rate when changing the axis speed

As long as the control signal is present, the affected variable is changed with the defined velocity.

Adjustment limits

The variable can only be adjusted via the corresponding inputs within the determined limit values.

Exception:

If the initial value of the variable to be adjusted is not in the specified limits, the affected variable can only be adjusted in the direction of the target range. If a modulo value is specified, the affected variable can be adjusted in both directions. After reaching the target range, it is no longer possible to leave this range.

Preset

The output variable can be pre-assigned to a definite value using the inputs "Preset" and "PresetValue". The "PresetAck" is set as soon as the "PresetValue" was applied. The output variable remains on the "PresetValue" as long as the "Preset" input is set.

Basic rules

- If both control signals ("Inc"/"Dec") are TRUE at the same time, they are evaluated as FALSE
- A control signal changes the polarity and causes an immediate change of direction. The current adjustment is also not completed
- If the adjustment velocity is changed during a motion, the behavior of the following function blocks differ:

IL_ContinuousAdjustType01/IL_ContinuousAdjustType02 and
IL_IncrementalAdjustType01

The change becomes immediately effective for IL_ContinuousAdjustType01 and IL_ContinuousAdjustType02. In contrast, the motion for IL_IncrementalAdjustType01 is finished before the new step velocity is accepted. A subsequent adjustment is performed with the adjusted variables.

Furthermore, the following rules apply to the function block IL_IncrementalAdjustType01:

- The inputs "Inc" and "Dec" are evaluated with regard to the edges. A rising edge causes an adjustment by the specified step width ("Step-Width")
- A new control signal (positive edge) with the same polarity detected during a running motion causes another motion following the current motion

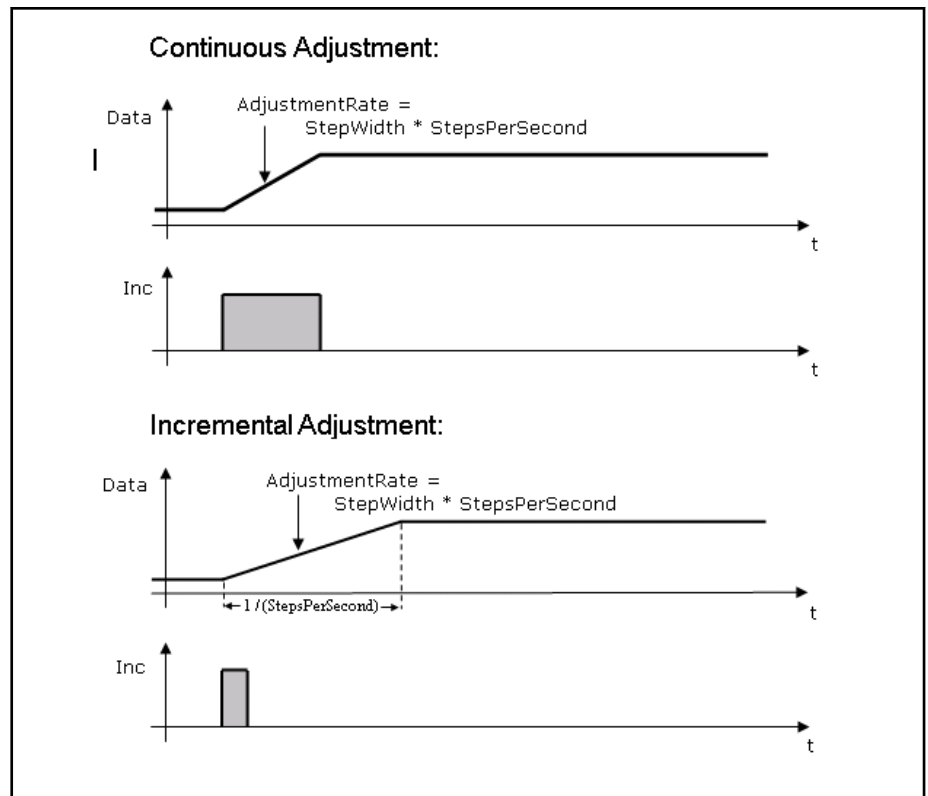


Fig. 4-3: Comparison of the behavior between a continuous and an incremental change

During the first activation ("Enable" = TRUE), the function blocks IL_ContinuousAdjustType01, IL_ContinuousAdjustType02 and IL_IncrementalAdjustType01 initialize the variables required to determine the cycle time (irrespective of the present inputs). In the second cycle, a calculated cycle time is available required to calculate the correct change rate. Subsequently, the cycle time is updated continuously.

By specifying a modulo value, the function block provides an output variable between zero and the specified modulo value. When exceeding the modulo value, the output variable is reset to zero. However, the adjustment continues. The specified valid range can thus be between zero and the modulo value. The range can be adjusted by the zero value.

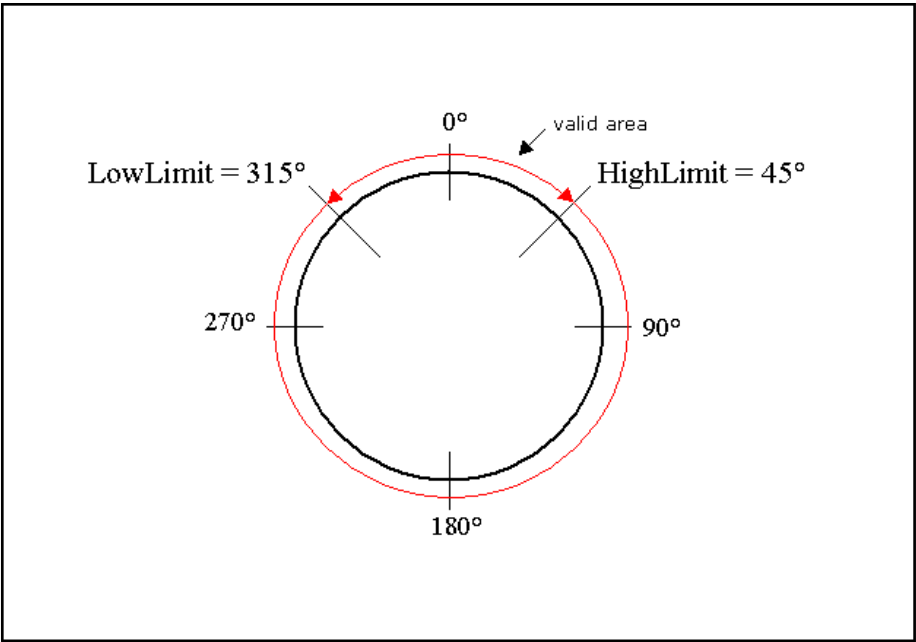


Fig. 4-4: Possible allocation of a valid range when specifying a modulo value of 360°

In addition, the function block IL_ContinuousAdjustType01 or IL_ContinuousAdjustType02 provides the possibility to adjust the change velocity depending on the time by specifying scaling factors and time intervals.

4.2.2 IL_ContinuousAdjustType01

Brief description The IL_ContinuousAdjustType01 function block allows a continuous adjustment of a REAL variable via binary inputs.

 In 14VRS, The "MB_ContinuousAdjustType01 (ML_TechBase library)" was renamed to "IL_ContinuousAdjustType01".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-3: Reference table of the IL_ContinuousAdjustType01 function block

Interface description

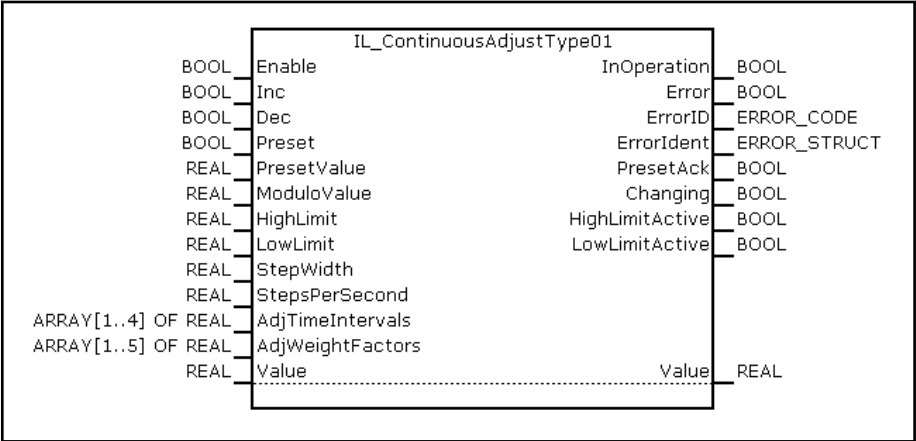


Fig. 4-5: IL_ContinuousAdjustType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Enabling the function block (cyclic, state-controlled)
	Inc	BOOL	Increase affected variable
	Dec	BOOL	Decrease affected variable
	Preset	BOOL	Set the preset value (PresetValue)
	PresetValue	REAL	Specified value for the preset value
	ModuloValue	REAL	Modulo value (if 0, then absolute processing)
	HighLimit	REAL	Maximum output value of the variable to be affected
	LowLimit	REAL	Minimum output value of the variable to be affected
	StepWidth	REAL	Step width
	StepsPerSecond	REAL	Number of steps per second
	AdjTimeIntervals	ARRAY [1..4] OF REAL	Time intervals in seconds to scale the change rate
	AdjWeightFactors	ARRAY [1..5] OF REAL	Scaling factors of the change rate
VAR_OUTPUT	InOperation	BOOL	Function block enabled. The outputs "HighLimitActive" and "LowLimitActive" are valid
	Error	BOOL	Calculation of the variable "Value" to be affected was completed with an error. The output variable "ErrorIdent" is valid
	ErrorID	ERROR_CODE	Brief error description
	ErrorIdent	ERROR_STRUCT	Detailed description of the diagnostics in the event of an error Troubleshooting, page 75
	PresetAck	BOOL	The value of the "PresetValue" input was applied
	Changing	BOOL	The variable to be affected was calculated. Output variable "Value" is valid
	HighLimitActive	BOOL	Maximum output value of the variable to be affected was reached
	LowLimitActive	BOOL	Minimum output value of the variable to be affected was reached
VAR_IN_OUT	Value	REAL	Variable to be affected

Tab. 4-4: Interface variables of the IL_ContinuousAdjustType01 function block

Signal-time diagram

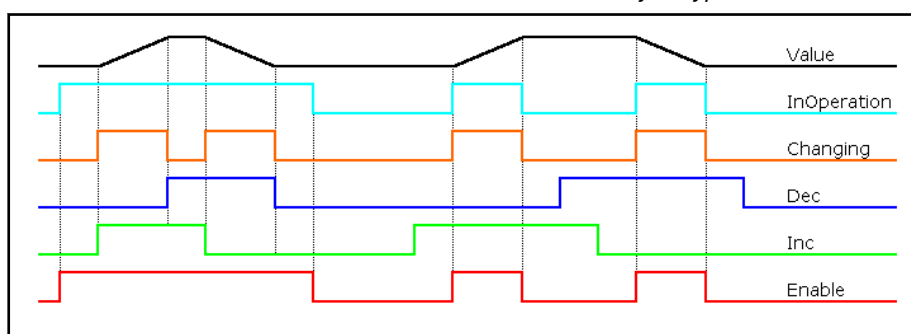


Fig. 4-6: Continuous parameter adjustment

Functional description The affected variable "Value" can be adjusted either in positive or negative direction. A valid range is specified to affect the variable using the limit values. The modulo value specification allows the adjustment in the reference system of the axis. Specify the step width and the number of steps per second to determine the change rate of the affected variable. Additionally, a specified value ("PresetValue") in the permissible operating range can be directly written to the affected variable.

After processing enabled via "Enable", the variable to be affected can be changed via the inputs "Inc" (in positive direction) and "Dec" (in negative direction). As long the "Inc" input or the "Dec" input is set, the adjustable variable can be continuously increased or decreased (" $\text{StepsPerSecond} \cdot \text{StepWidth}$ ").

When initializing the function block, ensure that the input variables "ModuloValue", "HighLimit" and "LowLimit" are provided with valid values.

Specify the scaling factors ("AdjWeightFactors[]") and their respective time intervals ("AdjTimeIntervals[]") to modify the change rate with regard to time. The scaling factors and the time intervals have to be set in addition to the inputs "StepsPerSecond" and "StepWidth". Depending on the current time interval, the change rate is multiplied with the respective scaling factor. The figure below shows the behavior of the function block using an example configuration: The "AdjTimeIntervals" input variables ([T1; T2; T3; T4]) are specified in seconds. T5 is the time interval between the completion of the time interval T4 and the adjustment abort. The "AdjWeightFactors" input variables ([0,5; 1; 1,5; 2; 4]) determine the scaling of the change rate. The last "AdjWeightFactors" value ("AdjWeightFactors [5]") is assigned to the time interval T5.

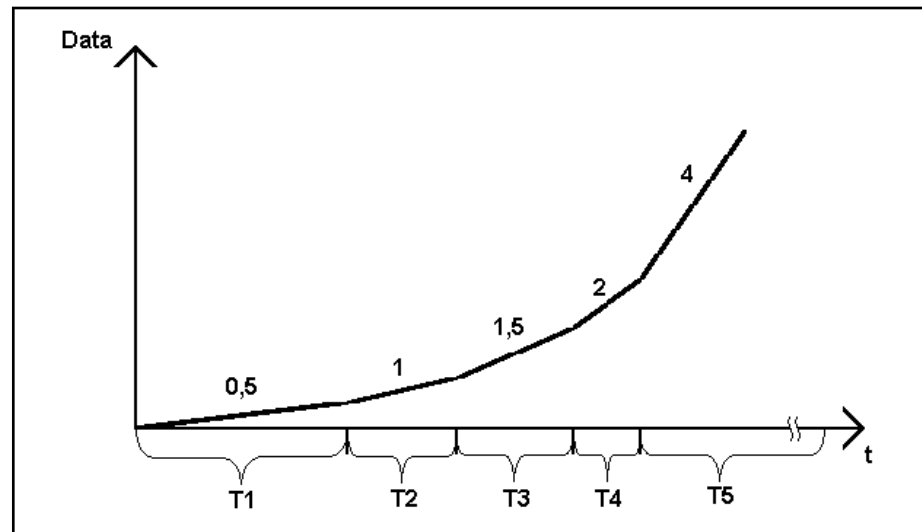


Fig. 4-7: Scaled adjustment



If a time interval of 0 is selected, the following scaling factor is retained until the adjustment is aborted.

Example:

If "AdjTimeIntervals[2]" = 0, the scaling factor of "AdjWeightFactors[3]" is retained until the process is aborted.

The calculation of the elapsed time to define the correct time interval of the scaling factors is reset with each new adjustment. If the inputs "AdjWeightFactors" and "AdjTimeIntervals" are not set, it is continuously adjusted with the change rate specified in the inputs "StepsPerSecond" and "StepWidth".



Positive values have to be specified for the input values of the time intervals ("AdjTimeIntervals"). The scaling factors can also contain negative values.

The variable to be affected can only be set to the specified preset value if it is in the valid range specified by "HighLimit" and "LowLimit".

The "InOperation" output signals that the function block is operating. However, there is no adjustment. The "Changing" output indicates that the adjustment was processed without errors. The value of the adjusted variable "Value" is valid.

The "Error" output indicates if an error occurs during the adjustment. For detailed information on the error, refer to the "ErrorIdent" structure.

Troubleshooting

The function block generates the following error messages in Additional1/Additional2 of the table **F_RELATED_TABLE**, 16#0170:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR (16#0006)	16#0002	16#0000	Inputs are not in the valid range
RESOURCE_ERROR (16#0003)	16#0004	16#0000	Drive firmware is not supported
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state

Tab. 4-5: Error codes of the IL_ContinuousAdjustType01 function block

4.2.3 IL_ContinuousAdjustType02

Brief description

The IL_ContinuousAdjustType02 function block allows a continuous adjustment of a DINT variable to be affected via binary inputs.



In 14VRS, The "MB_ContinuousAdjustType02 (ML_TechBase library)" was renamed to "IL_ContinuousAdjustType02".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-6: Reference table of the IL_ContinuousAdjustType02 function block

Interface description

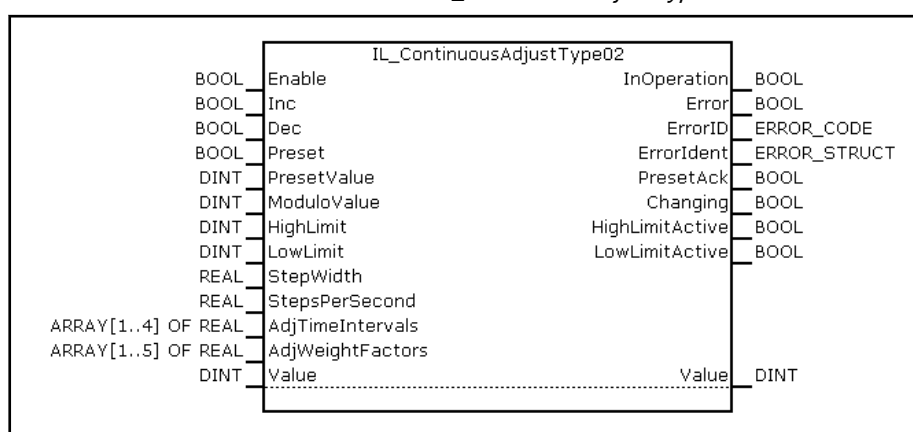


Fig. 4-8: IL_ContinuousAdjustType02 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Enabling the function block (cyclic, state-controlled)
	Inc	BOOL	Increase affected variable
	Dec	BOOL	Decrease affected variable
	Preset	BOOL	Set the preset value (PresetValue)
	PresetValue	DINT	Specified value for the preset value
	ModuloValue	DINT	Modulo value (if 0, then absolute processing)
	HighLimit	DINT	Maximum output value of the variable to be affected
	LowLimit	DINT	Minimum output value of the variable to be affected
	StepWidth	REAL	Step width
	StepsPerSecond	REAL	Steps per second
	AdjTimeIntervals	ARRAY [1..4] OF REAL	Time intervals in seconds to scale the change rate
	AdjWeightFactors	ARRAY [1..5] OF REAL	Scaling factors of the change rate
VAR_OUTPUT	InOperation	BOOL	Function block enabled. The outputs "HighLimitActive" and "LowLimitActive" are valid
	Error	BOOL	Calculation of the variable "Value" to be affected was completed with an error. The output variable "ErrorIdent" is valid
	ErrorID	ERROR_CODE	Brief error description
	ErrorIdent	ERROR_STRUCT	Detailed description of the diagnostics in the event of an error Troubleshooting, page 78
	PresetAck	BOOL	The value of the "PresetValue" input was applied
	Changing	BOOL	The variable to be affected was calculated. The output variable "Value" is valid
	HighLimitActive	BOOL	Maximum output value of the variable to be affected was reached
	LowLimitActive	BOOL	Minimum output value of the variable to be affected was reached
VAR_IN_OUT	Value	DINT	Variable to be affected

Tab. 4-7: Interface variables of the IL_ContinuousAdjustType02 function block

Signal-time diagram

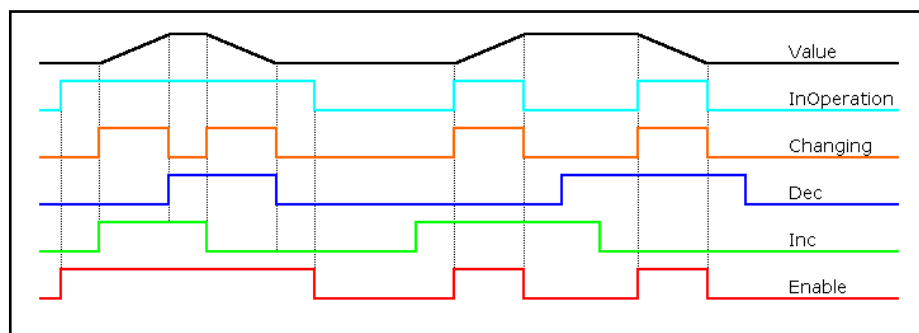


Fig. 4-9: Continuous parameter adjustment

Functional description

The affected variable "Value" can be adjusted either in positive or negative direction. A valid range is specified to affect the variable using the limit values. The modulo value specification allows the adjustment in the reference system of the axis. Specify the step width and the number of steps per second to determine the change rate of the affected variable. Additionally, a specified value (PresetValue) in the permissible operating range can be written directly to the affected variable.

After processing enabled via "Enable", the variable to be affected can be changed via the inputs "Inc" (in positive direction) and "Dec" (in negative direction). As long the "Inc" input or the "Dec" input is set, the adjustable variable can be continuously increased or decreased (" $\text{StepsPerSecond} \cdot \text{StepWidth}$ ").

When initializing the function block, ensure that the input variables "Modulo-Value", "HighLimit" and "LowLimit" are provided with valid values.

Specify the scaling factors ("AdjWeightFactors[]") and their respective time intervals ("AdjTimeIntervals[]") to modify the change rate with regard to time. The scaling factors and the time intervals have to be set in addition to the inputs "StepsPerSecond" and "StepWidth". Depending on the current time interval, the change rate is multiplied with the respective scaling factor. The figure below shows the behavior of the function block using an example configuration: The "AdjTimeIntervals" input variables (T_1 ; T_2 ; T_3 ; T_4) are specified in seconds. T_5 is the time interval between the completion of the time interval T_4 and the adjustment abort. The "AdjWeightFactors" input variables ($0,5$; 1 ; $1,5$; 2 ; 4) determine the scaling of the change rate. The last "AdjWeightFactors" value ("AdjWeightFactors[5]") is assigned to the time interval T_5 .

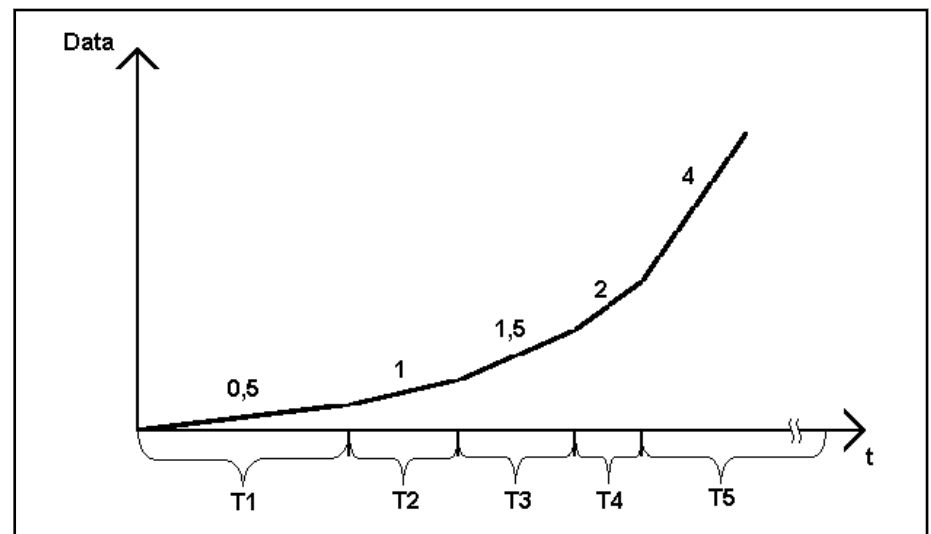


Fig. 4-10: Scaled adjustment



If a time interval of 0 is selected, the following scaling factor is retained until the adjustment is aborted.

Example:

If "AdjTimeIntervals[2]" = 0, the scaling factor of "AdjWeightFactors[3]" is retained until the process is aborted.

The calculation of the elapsed time to define the correct time interval of the scaling factors is reset with each new adjustment. If the inputs "AdjWeightFactors" and "AdjTimeIntervals" are not set, it is continuously adjusted with the change rate specified in the inputs "StepsPerSecond" and "StepWidth".



Positive values have to be specified for the input values of the time intervals (AdjTimeIntervals). The scaling factors can also contain negative values.

The variable to be affected can only be set to the specified preset value if it is in the valid range specified by "HighLimit" and "LowLimit".

The "InOperation" output signals that the function block is operating. However, there is no adjustment. The "Changing" output indicates that the adjustment was processed without errors. The value of the adjusted variable "Value" is valid.

The "Error" output indicates if an error occurs during the adjustment. For detailed information on the error, refer to the "ErrorIdent" structure.

Troubleshooting

The function block generates the following error messages in Additional1/Additional2 of the table "F_RELATED_TABLE", 16#0170:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR (16#0006)	16#0002	16#0000	Inputs are not in the valid range
RESOURCE_ERROR (16#0003)	16#0004	16#0000	Drive firmware is not supported
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state

Tab. 4-8: Error codes of the IL_ContinuousAdjustType02 function block

4.2.4 IL_IncrementalAdjustType01

Brief description

The IL_IncrementalAdjustType01 function block allows an incremental adjustment of a REAL variable via binary inputs.



In 14VRS, the "MB_IncrementalAdjustType01 (ML_TechBase library)" function block was renamed to "IL_IncrementalAdjustType01".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-9: Reference table of the IL_IncrementalAdjustType01 function block

Interface description

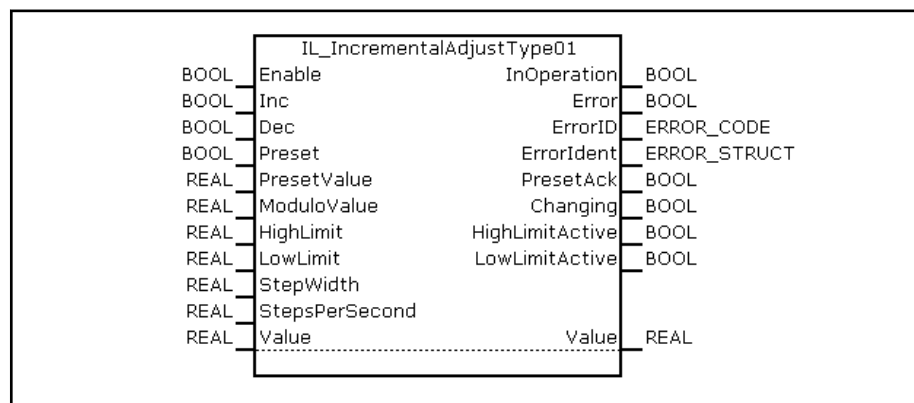


Fig. 4-11: IL_IncrementalAdjustType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Enabling the function block (cyclic, state-controlled)
	Inc	BOOL	Increase affected variable
	Dec	BOOL	Decrease affected variable
	Preset	BOOL	Set the preset value (PresetValue)
	PresetValue	REAL	Specified value for the preset value
	ModuloValue	REAL	Modulo value for rotary calculation
	HighLimit	REAL	Maximum output value of the variable to be affected
	LowLimit	REAL	Minimum output value of the variable to be affected
	StepWidth	REAL	Step width
	StepsPerSecond	REAL	Steps per second
VAR_OUTPUT	InOperation	BOOL	Function block enabled. The outputs "HighLimitActive" and "LowLimitActive" are valid
	Error	BOOL	Calculation of the variable "Value" to be affected was completed with an error. The output variable "ErrorIdent" is valid
	ErrorID	ERROR_CODE	Brief error description
	ErrorIdent	ERROR_STRUCT	Detailed description of the diagnostics in the event of an error
	PresetAck	BOOL	The value of the "PresetValue" input was applied
	Changing	BOOL	The variable to be affected was calculated. Output variable "Value" is valid
	HighLimitActive	BOOL	Maximum output value of the variable to be affected was reached
	LowLimitActive	BOOL	Minimum output value of the variable to be affected was reached
VAR_IN_OUT	Value	REAL	Variable to be affected

Tab. 4-10: Interface variables of the IL_IncrementalAdjustType01 function block

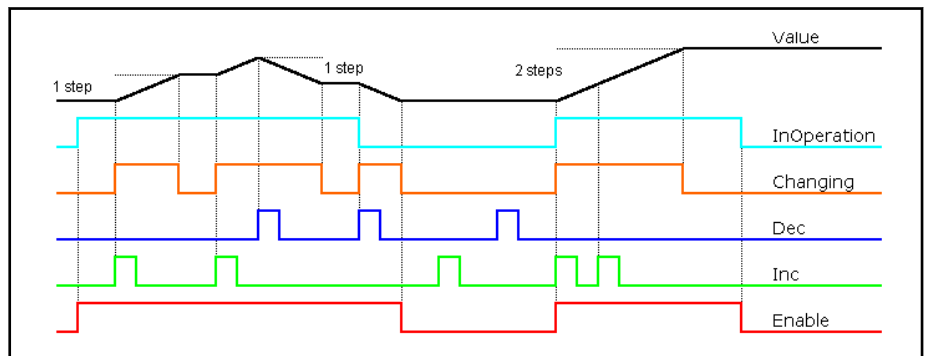
Signal-time diagram

Fig. 4-12: Incremental parameter adjustment

Functional description

The affected variable "Value" can be adjusted either in positive or negative direction. A valid range is specified to affect the variable using the limit values. The modulo value specification allows the adjustment in the reference system of the axis. Specify the step width and the number of steps per sec-

ond to determine the change rate of the affected variable. Additionally, a specified value ("PresetValue") in the permissible operating range can be directly written to the affected variable.

After processing enabled via "Enable", the variable to be affected can be changed via the inputs "Inc" (in positive direction) and "Dec" (in negative direction). As long the "Inc" input or the "Dec" input is set, the adjustable variable can be continuously increased or decreased ("StepsPerSecond" * "Step-Width").



The variable to be affected can only be set to the specified preset value if it is in the valid range specified by "HighLimit" and "LowLimit".

The "InOperation" output signals that the function block is operating. However, there is no adjustment. The "Changing" output indicates that the adjustment was processed without errors. The value of the adjusted variable "Value" is valid.

If an error occurs during the function block processing, this is signaled by the output "Error" = TRUE. For detailed information on the error, refer to the "ErrorIdent" output structure.

Troubleshooting

The function block generates the following error messages in Additional1/Additional2 of the table "F_RELATED_TABLE", 16#0170:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR (16#0006)	16#0002	16#0000	Inputs are not in the valid range
RESOURCE_ERROR (16#0003)	16#0004	16#0000	Drive firmware is not supported
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state

Tab. 4-11: Error codes of the IL_IncrementalAdjustType01 function block

4.2.5 Parameterizing function blocks to jog variables

Required hardware The following Rexroth hardware components are required:

- MLC hardware

Required software The following PC software has to be used:

- IndraWorks MLC 15VRS

Parameterizing the function blocks to adjust variables No special parameterization is required.

4.3 Basic classes for the function block development (Base FB)

4.3.1 Introduction and overview

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Basic classes for the function block development (Base FB)	
Basic function block IL_EnableDoneBase, page 86	Implements the interface of a function block with an "Enable" input and "Done", "Active" and "Standard" error outputs
Basic function block IL_EnableInOpBase, page 89	Implements the interface of a function block with an "Enable" input and "InOperation" and "Standard" error outputs

Basic function block IL_EnableInOpShutdown-Base, page 91	Implements the interface of a function block with an "Enable" input and "InOperation", "Shutdown" and "Standard" error outputs.
Basic function block IL_ExecuteDoneBase, page 94	Implements the interface of a function block with an "Execute" input and "Done", "Active" and "Standard" error outputs
Basic function block IL_ExecutelnOpBase, page 97	Implements the interface of a function block with an "Execute" input and "InOperation", "Active" and "Standard" error outputs
Basic function block IL_ExecuteDoneAborted-Base, page 100	Implements the interface of a function block with an "Execute" input and the "Done", "Active", "CommandAborted" and "Standard" error outputs
Basis function block IL_ExecuteDoneAborted-ShutdownBase, page 102	Implements the interface of a function block with an "Execute" input and "Done", "Active", "Shutdown", "CommandAborted" and "Standard" error outputs

Tab. 4-12: RIL_Uilities, Base FBs

The basic function blocks

- IL_EnableDoneBase
- IL_EnableInOpBase
- IL_EnableInOpShutdownBase
- IL_ExecuteDoneBase
- IL_ExecutelnOpBase
- IL_ExecuteDoneAbortedBase
- IL_ExecuteDoneAbortedShutdownBase

facilitate the development of function blocks. They implement the basic structure of the function block and the behavior of the standard inputs and outputs.

A state machine is specified for each of the two function block types which updates the standard inputs and outputs. The user code can be added to the respective state machine by implementing namely specified methods. Auxiliary methods are implemented facilitating the setting of error codes.

To use a basic function block, it has to be referenced via the keyword "EXTENDS" when defining the own function block. This procedure is also called "deriving a function block". The user-specific function block then contains the inputs and outputs of the basic function block and, subsequently, all inputs and outputs of the user-specific function block. All methods implemented in the basic function block can be used in the own (derived) function block. This is also called inheritance of methods.

Methods that can be overwritten

The basic function blocks define their own methods called by the internal state machine. These methods can be redefined to implement a user-specific functionality. This is also called overwriting of methods.

All methods of the basic function blocks that can be overwritten, are listed in the following.

Method	Description
mReset()	This method allows resetting of internal function blocks and self-defined outputs. It is called upon transition to the STATE_IDLE state. The outputs "Active", "In-Operation" and "Error" are reset internally. This method is to be implemented optionally.
mInit(Reset: BOOL): BOOL	Using this method, the preparations for the actual task of the function block are implemented. This method is called in or at the transition to the STATE_INIT state. "Reset" is TRUE during transition. The function block is initialized in mInit(). Until the initialization is completed, mInit() has to return TRUE. Once initialization is completed, mInit() has to return with FALSE. A subordinate state machine can be implemented in mInit(). This method is to be implemented optionally.
mInOp(Reset: BOOL): BOOL	The actual task of the function block is implemented in this method. This method is called in or at the transition to the STATE_INOP state. Argument "Reset" is TRUE during transition into STATE_INOP. For a function block with an "Execute" input, mInOp(FALSE) is called until FALSE is returned. Subsequently, STATE_DONE is automatically enabled if the function block is a derivation of a function block with a "Done" output.
mError()	If an error occurs, this method is called by mSetError(). Boolean outputs can be reset here. "Active", "InOperation" and the error outputs are already provided by mSetError(). This method is to be implemented optionally.
mIdle()	This method is called in the states STATE_IDLE, STATE_INIT and STATE_DONE and STATE_ERROR, but not in the STATE_INOP state. Function blocks provided with a continuous time measurement always have to be called. They can be called here.
mShutdown(Reset: BOOL): BOOL	Using this method, the end process of the function block is implemented. This method is called in or at the transition to the STATE_SHUTDOWN or STATE_ERROR state. The argument "Reset" is TRUE during the transition. As long as the end process is not completed, mShutdown() has to be called and TRUE returned. Once the end process is completed, mShutdown() has to return FALSE.

Auxiliary functions Auxiliary functions are methods that are provided by the basic function blocks and can be used in user-specific implementations. They may not be overwritten by user-specific implementations. Presently, two methods are implemented to set the error state.

Method	Description
mSetError(enErrorID: ERROR_CODE, enTable: ERROR_TABLE, dwAdditional1: DWORD, dwAdditional2: DWORD)	This method sets the specified error code at the outputs "ErrorID" and "ErrorIdnt" and activates the output "Error". Subsequently, mError() is called. The function block changes to STATE_ERROR. After calling mSetError(), a return jump should be performed using RETURN to impede further code processing after an error.
mSetErrorIdnt(enErrorID: ERROR_CODE, stErrorIdnt: ERROR_STRUCT)	This method sets the specified error code at the outputs "ErrorID" and "ErrorIdnt" and activates the output "Error". Subsequently, mError() is called. The function block changes to STATE_ERROR. This function is suitable for applying the error code from another function block. After calling mSetErrorIdnt(), a return jump should be performed using RETURN to impede further code processing after an error.
mSetCommandAborted()	This method is only available for the basic function block IL_ExecuteDoneAbortedBase. Calling this method sets the "CommandAborted" output and switches the function blocks to the "CommandAborted" state.

Using the basic function blocks

Basic function blocks can be extended to derive user-specific function blocks. Therefore, the "Extended:" checkbox has to be selected and the basic function block has to be entered into the text box when the new function block is created.

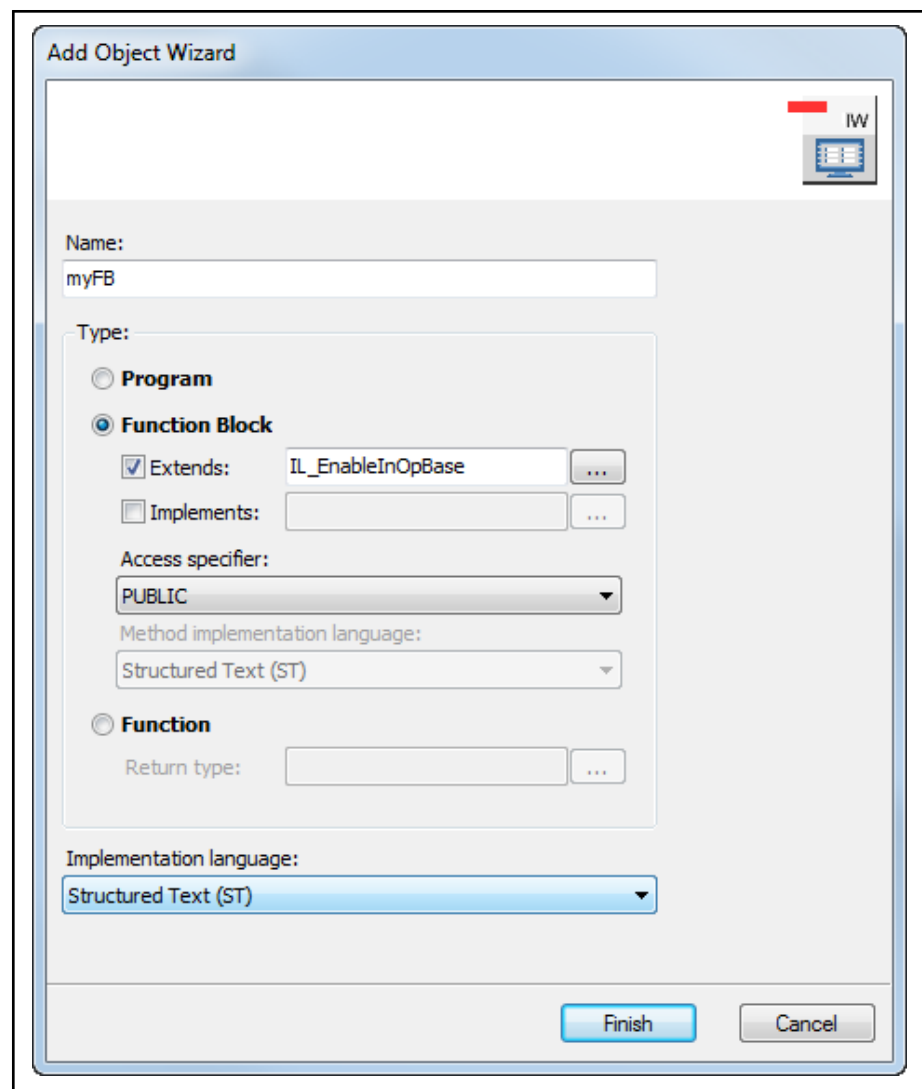


Fig. 4-13: Using a basic function block

Inheritance is the basic principle. The new function block contains all variables and methods already implemented in the basic function block. User-specific functionality can be added by re-implementing existing methods. The newly declared inputs and outputs in the function block derived from the basic block are displayed below the interfaces already defined in the basic function block. It is therefore only necessary to define the new inputs and outputs.

It is mandatory to call "FB_Main()" in the implementation section of the derived function block. Other implementations are not allowed at this point.

```

1  FUNCTION_BLOCK myFB EXTENDS IL_EnableInOpBase
2  VAR_INPUT
3      MyInput: REAL;
4  END_VAR
5  VAR_OUTPUT
6      MyOutput: REAL;
7  END_VAR
8  VAR
9      iInitState: INT;
10 END_VAR
11
1  FB_Main();

```

Fig. 4-14: Implementation of a function block based on the basic function block *IL_EnableInOpBase*

Interface of the created function block:

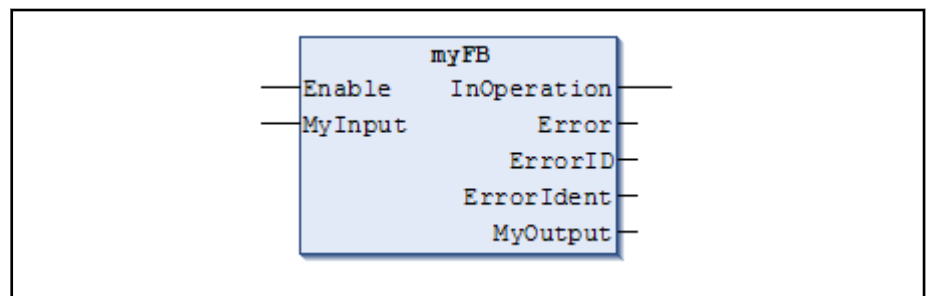


Fig. 4-15: Interface of the example function block based on *IL_EnableInOpBase*

The actual function is implemented in the rewritable methods "mInit()" and "mInOp()". Methods that are not required, do not have to be implemented.

The following figure shows the implementation of mInit() method. The state machine shown in the example is optional.

```

1  METHOD mInit : BOOL
2  VAR_INPUT
3      Reset: BOOL;
4  END_VAR
5  VAR_CONSTANT
6      // define error codes
7      ADD1_STATEMACHINE: DWORD := 16#00000006;
8      ADD2_STATEMACHINE: DWORD := 16#00000000;
9  END_VAR

1  IF Reset=TRUE THEN
2      // First call => entering Init state
3      iInitState := 0; // Reset state machine
4  END_IF
5
6  mInit := TRUE; // Call me again
7
8  CASE iInitState OF
9      0:
10         // Write your own code here
11         iInitState := iInitState + 1;
12      1:
13         // Write your own code here
14         mInit := FALSE; // Init finished => don't call me again
15  ELSE
16         // Undefined state => error
17         mSetError (STATE_MACHINE_ERROR,
18                 F_RELATED_TABLE,
19                 ADD1_STATEMACHINE,
20                 ADD2_STATEMACHINE);
21  END_CASE

```

Fig. 4-16: Implementation of an mInit() method

4.3.2 Basic function block IL_EnableDoneBase

Brief description

This basic function block (basic class) implements the interface of a function block with "Enable" input and "Done" and "Active" outputs. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

Function blocks of this type are mainly used for cyclic processes, such as the cyclic polling of an external device.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Interface description

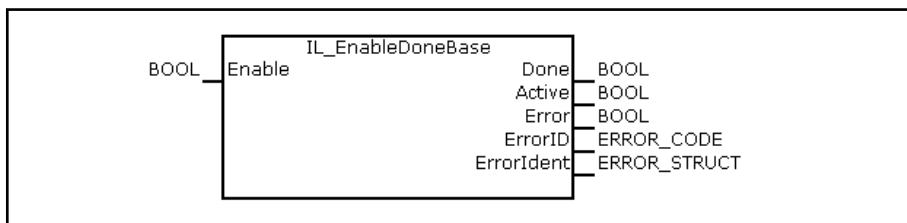


Fig. 4-17: Function block IL_EnableDoneBase

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing enabled (cyclic, state-controlled)
VAR_OUTPUT	Done	BOOL	The function block has output a result
	Active	BOOL	The function block is active
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-13: Function block IL_EnableDoneBase

Functional description

This basic class implements the interface of a function block with "Enable" input and "Done" as well as "Active" outputs. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Function blocks with Enable input are mainly used for tasks that are cyclically repeated.

This basic function block implements the "Enable" input and the "Done", "Active", "Error", "ErrorID" and "ErrorIdent" outputs. Processing starts at a rising edge at "Enable" and runs after the initialization phase until either "Enable" is reset or an error occurs. After the initialization phase, "Active" is set to indicate that the function block is performing its task. After completion of the task, "Done" is set for one cycle. Subsequently, a new processing procedure is started.

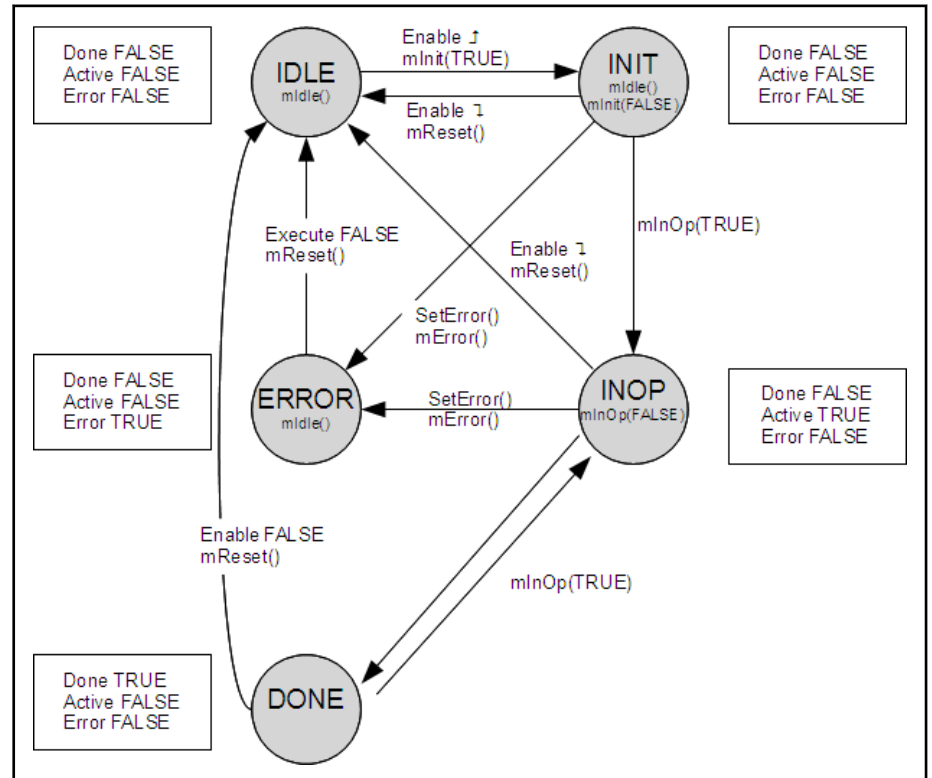


Fig. 4-18: State machine of a function block with Enable input and Done output

Function blocks of this type have five internal states (see also [IL_FB_STATES](#) on page 107):

STATE_IDLE	This state represents the starting state of the function block. It is activated when "Enable" is FALSE. mReset() is executed at the beginning of the state. mIdle() is executed in the state. If "Enable" is set, the STATE_INIT state is entered.
STATE_INIT	This state is used to start the function block. It runs before the actual cyclical operation. mInit(Reset:=TRUE) is called at the beginning of the state. mInit(Reset:=FALSE) and mIdle() are called in the state. If mInit() returns FALSE, the STATE_INOP state is entered. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Enable" is reset, the STATE_IDLE state is entered.
STATE_INOP	The cyclic task is performed here. The "Active" output is TRUE as long as this state is active. mInOp(Reset:=TRUE) is called at the beginning of the state. mInOp(Reset:=FALSE) is called in the state. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Enable" is reset, the STATE_IDLE state is entered.
STATE_DONE	State active after successful task processing. It is left after another cycle. Subsequently, STATE_INOP will become active again.
STATE_ERROR	This is the error state of the function block. Error output "Error" is active in this state and the error code is displayed at "Error-ID" or "ErrorIdent". mError() is called at the beginning of the state. mIdle() is called in the state. If "Enable" is reset, the STATE_IDLE state is entered.

Signal-time diagram The state machine described above yields the following timing behavior:

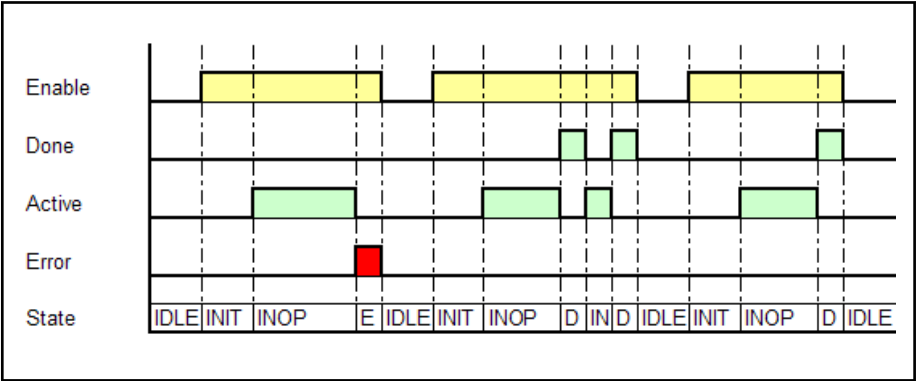


Fig. 4-19: Time response of a function block-based on IL_EnableDoneBase

Error codes The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

Tab. 4-14: Error codes generated by IL_EnableDoneBase

4.3.3 Basic function block IL_EnableInOpBase

Brief description This basic function block (basic class) implements the interface of a function block with "Enable" input and "InOperation" output. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

Function blocks of this type are mainly used for continuous processes, such as controllers.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Interface description

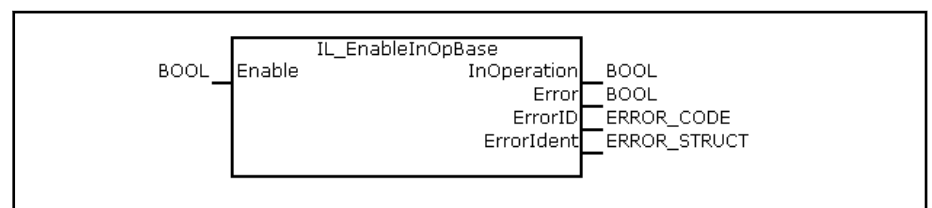


Fig. 4-20: Function block IL_EnableInOpBase

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing enabled (cyclic, state-controlled)
VAR_OUTPUT	InOperation	BOOL	The function blocks is busy with its actual task
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-15: Function block IL_EnableInOpBase

Functional description This basic class implements the interface of a function block with "Enable" input and "InOperation" output. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

This basic function block implements the "Enable" input and the "InOperation", "Error", "ErrorID" and "ErrorIdent" outputs. Processing starts at a rising edge at "Enable" and runs after the initialization phase until either "Enable" is reset or an error occurs. After completion of the initialization phase, "InOperation" is set to indicate that the function block is performing its cyclical task.

Function blocks with "Enable" input "InOperation" output are preferably used in tasks that are cyclically repeated, e.g., control.

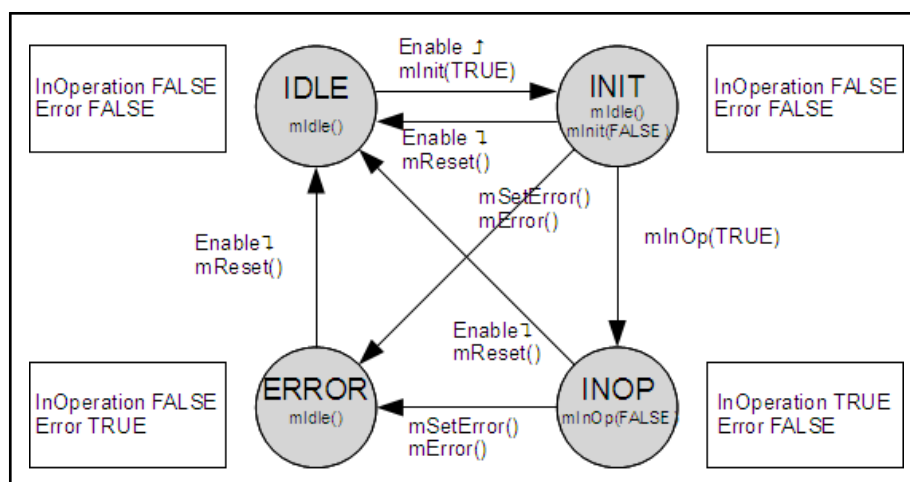


Fig. 4-21: State machine of a function block with Enable input and InOperation output

Function blocks of this type have four internal states (see also [IL_FB_STATES](#) on page 107):

- STATE_IDLE** This state represents the starting state of the function block. It is activated when "Enable" is FALSE. mReset() is executed at the beginning of the state. mIdle() is executed in the state. If "Enable" is set, the STATE_INIT state is entered.
- STATE_INIT** This state is used to start the function block. It runs before the actual cyclical operation. mInit(Reset:=TRUE) is called at the beginning of the state. mInit(Reset:=FALSE) and mIdle() are called in the state. If mInit() returns FALSE, the STATE_INOP state is entered. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Enable" is reset, the STATE_IDLE state is entered.
- STATE_INOP** The cyclic task is performed here. The "InOperation" output is TRUE as long as this state is active. mInOp(Reset:=TRUE) is called at the beginning of the state. mInOp(Reset:=FALSE) is called in the state. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Enable" is reset, the STATE_IDLE state is entered.
- STATE_ERROR** This is the error state of the function block. Error output "Error" is active in this state and the error code is displayed at "Error-ID" or "ErrorIdent". mError() is called at the beginning of the state. mIdle() is called in the state. If "Enable" is reset, the STATE_IDLE state is entered.

Signal-time diagram The state machine described above yields the following timing behavior:

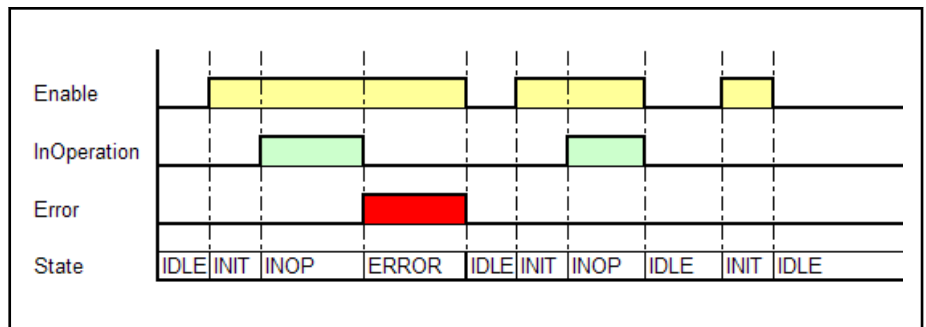


Fig. 4-22: Time response of a function block-based on IL_EnableInOpBase

Error codes The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

Tab. 4-16: Error codes generated by IL_EnableInOpBase

4.3.4 Basic function block IL_EnableInOpShutdownBase

Brief description This basic function block (basic class) implements the interface of a function block with "Enable" input and "InOperation" and "Shutdown" output. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

Function blocks of this type are mainly used for continuous processes, such as controllers.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Interface description

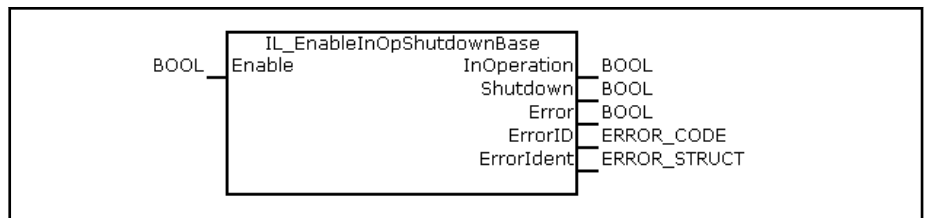


Fig. 4-23: Function block IL_EnableInOpShutdownBase

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing enabled (cyclic, state-controlled)
VAR_OUTPUT	InOperation	BOOL	The function blocks is busy with its actual task
	Shutdown	BOOL	Indicates that the function block is currently in a defined final state
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-17: Function block IL_EnableInOpShutdownBase

Functional description This basic class implements the interface of a function block with "Enable" input and "InOperation" and "Shutdown" output. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

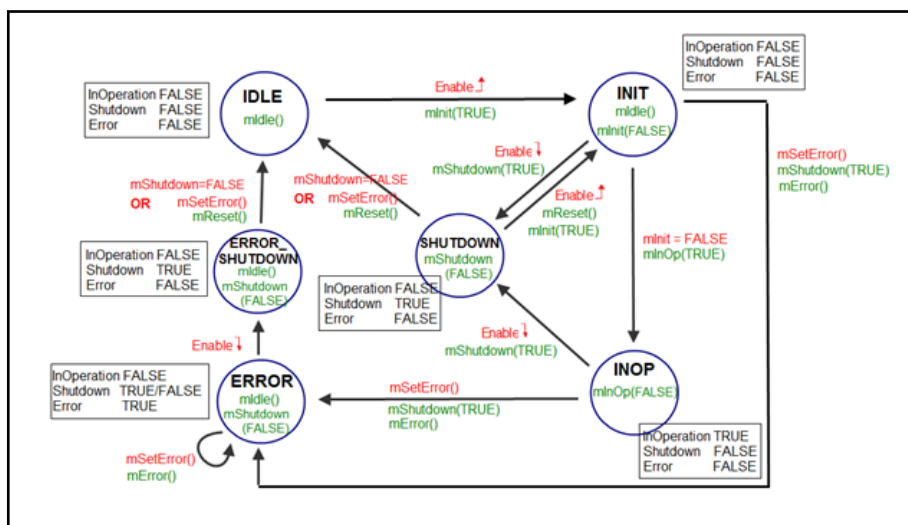
This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

This basic function block implements the "Enable" input and the "InOperation", "Shutdown", "Error", "ErrorID" and "ErrorIdent" outputs.

Processing starts at a rising edge at "Enable" and runs after the initialization phase until either "Enable" is reset or an error occurs. After completion of the initialization phase, "InOperation" is set to indicate that the function block is performing its cyclical task.

The "Shutdown" signalizes that the function block changes to state "SHUT-DOWN". As long as the "Shutdown" output is set, the function block has to be called.

Function blocks with "Enable" input "InOperation" output are preferably used in tasks that are cyclically repeated, e.g., control.



Red/top State switching conditions

Green/bottom Called methods during state switching

Fig. 4-24: State machine of a function block with Enable input and InOperation/shutdown output

Function blocks of this type have six internal states (see also [IL_FB_STATES](#) on page 107):

STATE_IDLE This state represents the starting state of the function block. It is activated when "Enable" is FALSE. mReset() is executed at the beginning of the state. mIdle() is executed in the state.

If "Enable" is set, the STATE_INIT state is entered.

STATE_INIT This state is used to start the function block. It runs before the actual cyclical operation. mInit(Reset:=TRUE) is called at the beginning of the state. mInit(Reset:=FALSE) and mIdle() are called in the state.

If mInit() returns FALSE, the STATE_INOP state is entered. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Enable" is reset, the STATE_SHUTDOWN state is entered.

STATE_INOP	The cyclic task is performed here. The "InOperation" output is TRUE as long as this state is active. mInOp(Reset:=TRUE) is called at the beginning of the state. mInOp(Reset:=FALSE) is called in the state. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Enable" is reset, the STATE_SHUTDOWN state is entered.
STATE_SHUT-DOWN	This is the final state of the function block. The "Shutdown" output is TRUE as long as this state is active. mShutDown(Reset:=TRUE) is called upon entry in the state. mShutDown(Reset:=FALSE) is called in the state. If mShutDown() is False, the STATE_IDLE state is entered.
STATE_ERROR	This is the error state of the function block. Error output "Error" is active in this state and the error code is displayed at "ErrorID" or "ErrorIdent". The shutdown process is executed in this state. mError() and mShutDown(Reset := TRUE) is called upon entry in the state. mIdle() and mShutDown(Reset := FALSE) is called in the state. As long as mShutDown() returns "TRUE", the output "Shutdown" remains "TRUE". If an error occurs during the shutdown process, the shutdown process is canceled and the "ShutDown" output is reset. If "Enable" is reset, the STATE_ERROR_SHUTDOWN state is entered.
STATE_ERROR_SHUTDOWN	The prerequisite to enter this state is that the previous state was "STATE_ERROR" and a falling edge at "Enable". If the shutdown is not yet completed, the program remains in this state and is called with mShutDown (reset := FALSE). The "ShutDown" output is "TRUE" as long as this state is active. If mShutDown() is False, the STATE_IDLE state is entered. If an error occurs during the shutdown operation, the operation is canceled. The "Error" output is set for one cycle. The "ShutDown" output is reset. The state machine changes to STATE_IDLE. If it was already shut down in STATE_ERROR, mShutDown() is not called in this state. It is immediately changed to STATE_IDLE.

Signal-time diagram The state machine described above yields the following timing behavior:

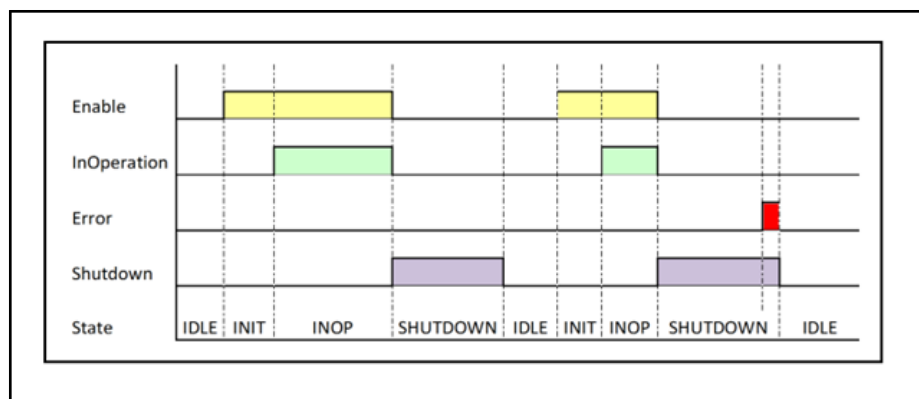


Fig. 4-25: Time response of a function block based on IL_EnableInOpShut-Base (I)

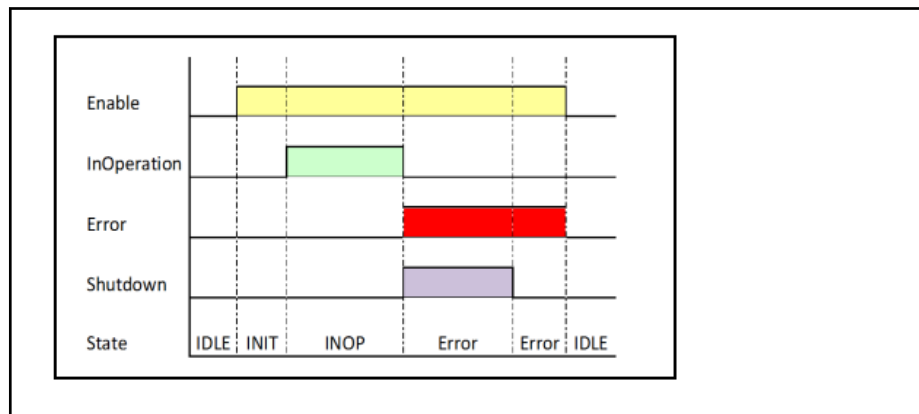


Fig. 4-26: Time response of a function block based on IL_EnableInOpShut-Base (II)

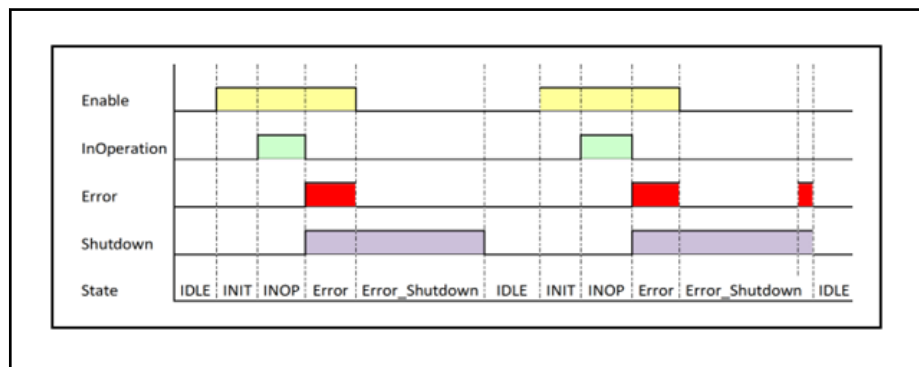


Fig. 4-27: Time response of a function block based on IL_EnableInOpShut-Base (III)

Error codes The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

Tab. 4-18: Error codes generated by IL_EnableInOpShutdownBase

4.3.5 Basic function block IL_ExecuteDoneBase

Brief description This basic function block (basic class) implements the interface of a function block with "Execute" input and "Done" and "Active" outputs. In addition, de-

fault error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

Function blocks of this type are used for nonrecurring tasks requiring a processing time of several cycles, for example, initialization of external hardware.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Interface description

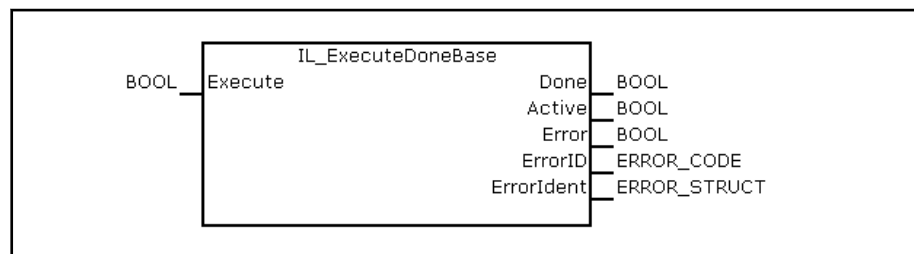


Fig. 4-28: IL_ExecuteDoneBase function block

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Processing enabled (cyclic, edge-controlled)
VAR_OUTPUT	Done	BOOL	Processing completed by the function block without error
	Active	BOOL	The function block was started
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-19: IL_ExecuteDoneBase function block

Functional description

This basic class implements the interface of a function block with "Execute" input and "Done" as well as "Active" outputs. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

This basic function block implements the "Execute" input and the "Done", "Active", "Error", "ErrorID" and "ErrorIdent" outputs.

Processing starts at a rising edge at "Execute" and runs after the initialization phase until either STATE_DONE is reached or an error occurs.

A new rising edge at "Execute" causes a reset and a restart of the function block.

Function blocks with an "Execute" input and a "Done" output are preferably used in nonrecurring tasks with a defined conclusion, e.g., initialization of external hardware.

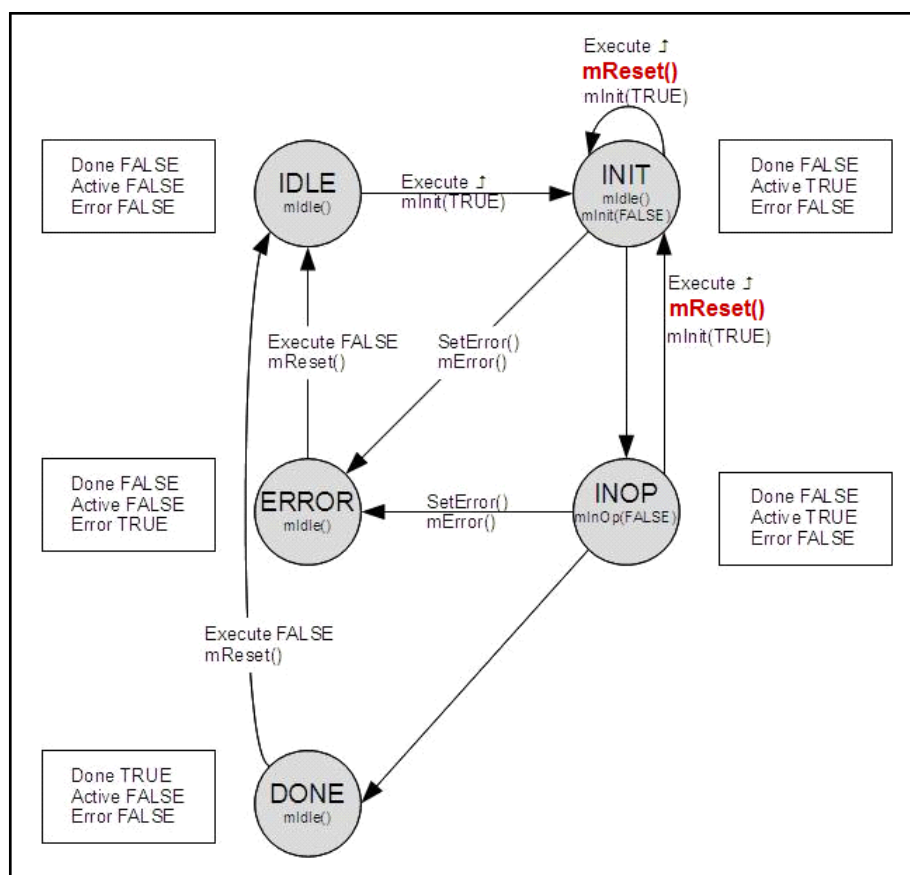


Fig. 4-29: State machine of a function block with Execute input and Done as well as Active outputs

Function blocks of this type have five internal states (see also [IL_FB_STATES](#) on page 107):

STATE_IDLE

This state represents the starting state of the function block. It is enabled if "Execute" is FALSE and if the function block is either in STATE_DONE or STATE_ERROR. mReset() is executed at the beginning of the state. mIdle() is executed in the state.

If "Execute" is set, the STATE_INIT state is entered.

STATE_INIT

This state is used to start the function block. It runs before the actual cyclical operation. The "Active" output is TRUE as long as this state is active. mInit(Reset:=TRUE) is called at the beginning of the state. mInit(Reset:=FALSE) and mIdle() are called in the state.

If mInit() returns FALSE, the STATE_INOP state is entered. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered.

STATE_INOP	The cyclic task is performed here. The "Active" output is TRUE as long as this state is active. mInOp(Reset:=TRUE) is called at the beginning of the state. mInOp(Reset:=FALSE) is called in the state. If mInOp() returns FALSE, the task is completed and the function block enters the STATE_DONE state. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered.
STATE_DONE	State active after successful task processing. It is exited if "Execute" is FALSE but remains active for at least one cycle. mIdle() is called in the state.
STATE_ERROR	This is the error state of the function block. Error output "Error" is active in this state and the error code is displayed at "Error-ID" or "ErrorIdent". mError() is called at the beginning of the state. mIdle() is called in the state. If "Execute" is reset, the STATE_IDLE state is entered.

Signal-time diagram

The state machine described above yields the following timing behavior:

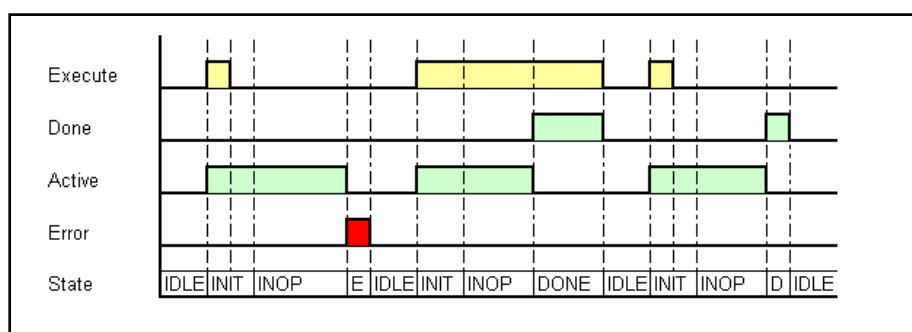


Fig. 4-30: Time response of a function block-based on IL_ExecuteDoneBase

Error codes

The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

Tab. 4-20: Error codes generated by IL_ExecuteDoneBase

4.3.6 Basic function block IL_ExecutelnOpBase

Brief description

This basic function block (basic class) implements the interface of a function block with "Execute" input and "InOperation" and "Active" output. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

Function blocks of this type are used for constant tasks without defined termination.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Interface description

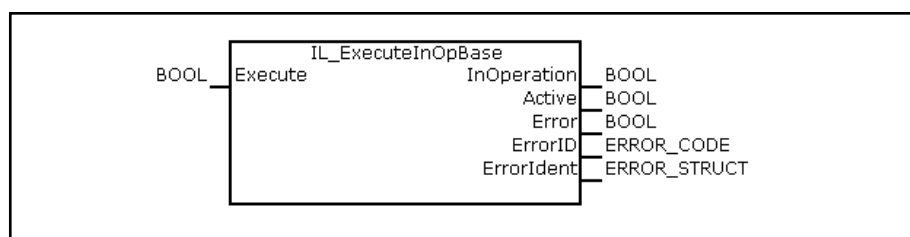


Fig. 4-31: Function block IL_ExecuteInOpBase

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Processing enabled (cyclic, edge-controlled)
VAR_OUTPUT	InOperation	BOOL	The function block processes its task
	Active	BOOL	The function block was started
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-21: Function block IL_ExecuteInOpBase

Functional description

This basic class implements the interface of a function block with "Execute" input and "InOperation" as well as "Active" outputs. In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

This basic function block implements the "Execute" input and the "InOperation", "Active", "Error", "ErrorID" and "ErrorIdent" outputs. Processing starts with a rising edge at "Execute" and runs after the initialization phase until an error occurs. A new rising edge at "Execute" causes a reset and a restart of the function block.

Function blocks with "Execute" input "InOperation" output are preferably used for continuous tasks without defined completion such as closed-loop velocity control.

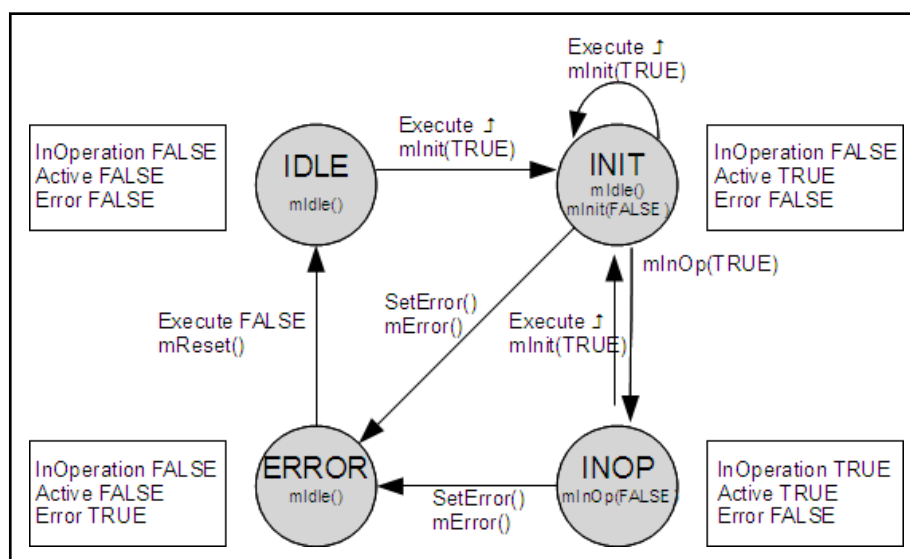


Fig. 4-32: State machine of a function block with Execute input and InOperation as well as Active outputs

Function blocks of this type have four internal states (see also [IL_FB_STATES](#) on page 107):

- STATE_IDLE** State represents the starting state of the function block. It is enabled if "Execute" is FALSE and if the function block is in STATE_DONE. mReset() is executed at the beginning of the state. mIdle() is executed in the state.
If "Execute" is set, the STATE_INIT state is entered.
- STATE_INIT** This state is used to start the function block. It runs before the actual cyclical operation. The "Active" output is TRUE as long as this state is active. mInit(Reset:=TRUE) is called at the beginning of the state. mInit(Reset:=FALSE) and mIdle() are called in the state.
If mInit() returns FALSE, the STATE_INOP state is entered. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered.
- STATE_INOP** The cyclic task is performed here. The "Active" output is TRUE as long as this state is active. mInOp(Reset:=TRUE) is called at the beginning of the state. mInOp(Reset:=FALSE) is called in the state.
If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered.
- STATE_ERROR** This is the error state of the function block. Error output "Error" is active in this state and the error code is displayed at "Error-ID" or "ErrorIdent". mError() is called at the beginning of the state. mIdle() is called in the state.
If "Execute" is reset, the STATE_IDLE state is entered.

Signal-time diagram

The state machine described above yields the following timing behavior:

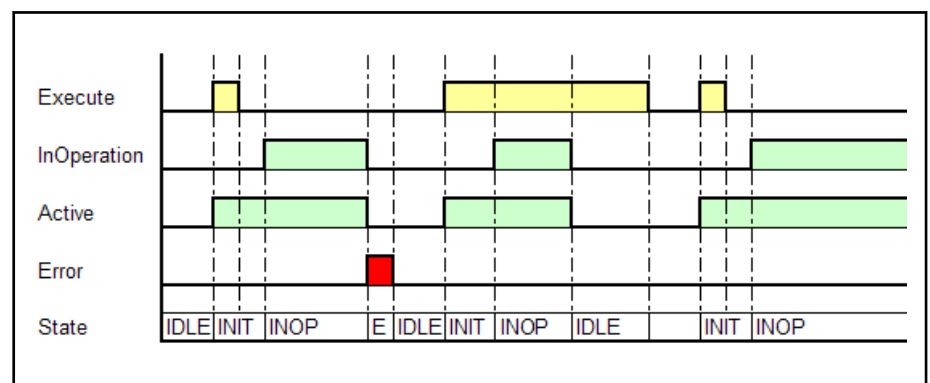


Fig. 4-33: Time behavior of a function block-based on IL_ExecutelnOpBase

Error codes

The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

Tab. 4-22: Error codes generated by IL_ExecutelnOpBase

4.3.7 Basic function block IL_ExecuteDoneAbortedBase

Brief description This basic function block (basic class) implements the interface of a function block with the "Execute" input and the outputs "Done", "CommandAborted" and "Active". In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

Function blocks of this type are used for nonrecurring tasks requiring a processing time of several cycles, for example, initialization of external hardware.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Interface description

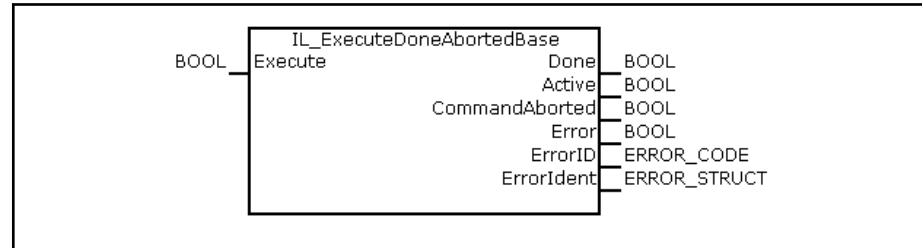


Fig. 4-34: IL_ExecuteDoneAbortedBase function block

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Processing enabled (cyclic, edge-controlled)
VAR_OUTPUT	Done	BOOL	Processing completed by the function block without error
	Active	BOOL	The function block was started
	CommandAborted	BOOL	The processing was aborted by another function block
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-23: IL_ExecuteDoneAbortedBase function block

Functional description This basic class implements the interface of a function block with the "Execute" input and the outputs "Done", "CommandAborted" and "Active". In addition, default error outputs are applied. The function block implements a state machine that is suitable for the default inputs and outputs.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via keyword "EXTENDS".

Function blocks with an "Execute" input are preferably used in nonrecurring tasks with a defined conclusion, e.g., initialization of external hardware. This basic function block implements the "Execute" input and the "Done", "Active", "Error", "ErrorID" and "ErrorIdent" outputs. Processing starts at a rising edge at "Execute" and runs after the initialization phase until either the STATE_DONE state is reached, an error occurs or the processing is aborted by another function block.

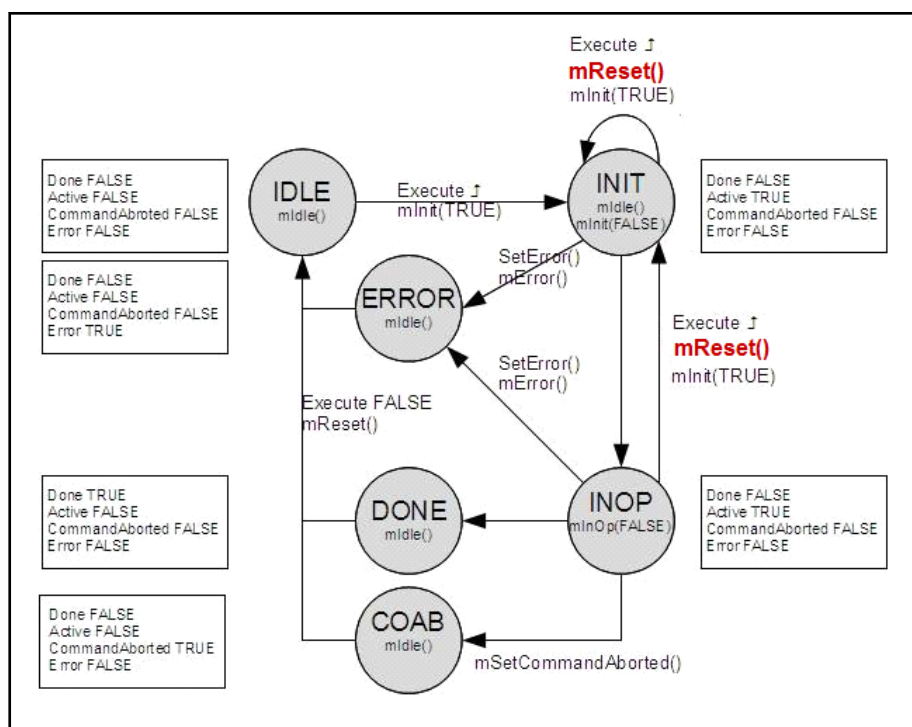


Fig. 4-35: State machine of a function block with the "Execute" input and the outputs "Done", "CommandAborted" as well as "Active"

Function blocks of this type have six internal states (see also [IL_FB_STATES](#) on page 107):

STATE_IDLE

State represents the starting state of the function block. It is enabled if "Execute" is FALSE and if the function block is either in STATE_DONE or STATE_ERROR. mReset() is executed at the beginning of the state. mIdle() is executed in the state.

If "Execute" is set, the STATE_INIT state is entered.

STATE_INIT

This state is used to start the function block. It runs before the actual cyclical operation. The "Active" output is TRUE as long as this state is active. mInit(Reset:=TRUE) is called at the beginning of the state. mInit(Reset:=FALSE) and mIdle() are called in the state.

If mInit() returns FALSE, the STATE_INOP state is entered. If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered. If "Execute" is reset, the STATE_IDLE state is entered.

STATE_INOP

The cyclic task is performed here. The "Active" output is TRUE as long as this state is active. mInOp(Reset:=TRUE) is called at the beginning of the state. mInOp(Reset:=FALSE) is called in the state.

If mInOp() returns FALSE, the task is completed and the function block enters the STATE_DONE state.

If mSetError() or mSetErrorIdent() is called, the STATE_ERROR state is entered.

- STATE_DONE

State active after successful task processing. It is exited if "Execute" is FALSE but remains active for at least one cycle. mIdle() is called in the state.
- STATE_COAB

State becomes active after the auxiliary method "mSetCommandAborted" was completely processed. It is exited if "Execute" is FALSE but remains active for at least one cycle. mIdle() is called in the state.
- STATE_ERROR

This is the error state of the function block. Error output "Error" is active in this state and the error code is displayed at "Error-ID" or "ErrorIdent". mError() is called at the beginning of the state. mIdle() is called in the state.

If "Execute" is reset, the STATE_IDLE state is entered.

Signal-time diagram The state machine described above yields the following timing behavior:

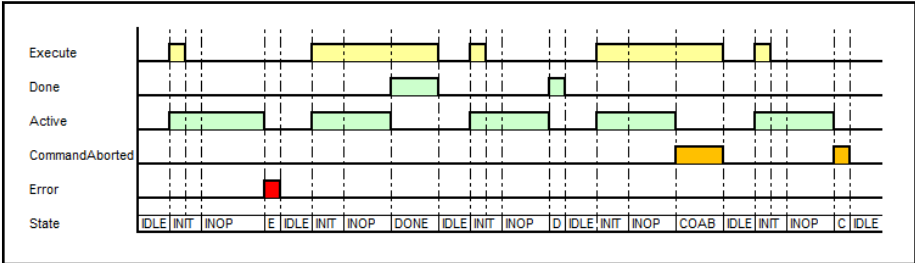


Fig. 4-36: Time response of a function block-based on IL_ExecuteDoneAbortedBase

Error codes The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

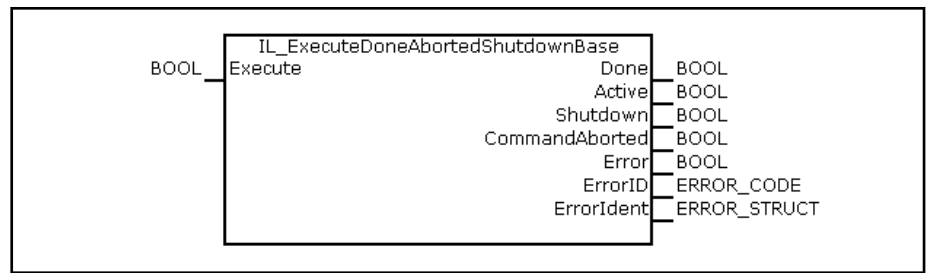
Tab. 4-24: Error codes generated by IL_ExecuteDoneAbortedBase

4.3.8 Basis function block IL_ExecuteDoneAbortedShutdownBase

Brief description This basic function block (basic class) implements the interface of a function block with "Execute" input and "Done", "Active", "Shutdown" and "CommandAborted" outputs. In addition, the default error outputs are created. The function block implements a state machine suitable for the default inputs and outputs.

Function blocks of this type are used for nonrecurring tasks requiring a processing time of multiple cycles. for example, the initialization of an external hardware.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via the keyword "EXTENDS".

Interface descriptionFig. 4-37: *IL_ExecuteDoneAbortedShutdownBase function block*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Processing enabled (cyclic, edge-controlled)
VAR_OUTPUT	Done	BOOL	Processing completed by the function block without error
	Active	BOOL	The function block was started
	Shutdown	BOOL	Processing is completed. It was either aborted by the user (mAbort()) or due to an error (mSetError() or mSetErrorIdent()). However, the function block has to successfully complete its activities
	CommandAborted	BOOL	The processing was aborted by another function block
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause

Tab. 4-25: *IL_ExecuteDoneAbortedShutdownBase function block***Functional description**

This basic class implements the interface of a function block with the "Execute" input and the "Done", "Active", "Shutdown" and "CommandAborted" outputs. In addition, the default error outputs are created.

The function block implements a state machine suitable for the default inputs and outputs.

This function block serves as a basic class for user-specific implementations. It has to be incorporated via the keyword "EXTENDS".

Processing starts at a rising edge at "Execute" and runs after the initialization phase until either STATE_DONE is reached, an error occurs or the processing is aborted by another function block.

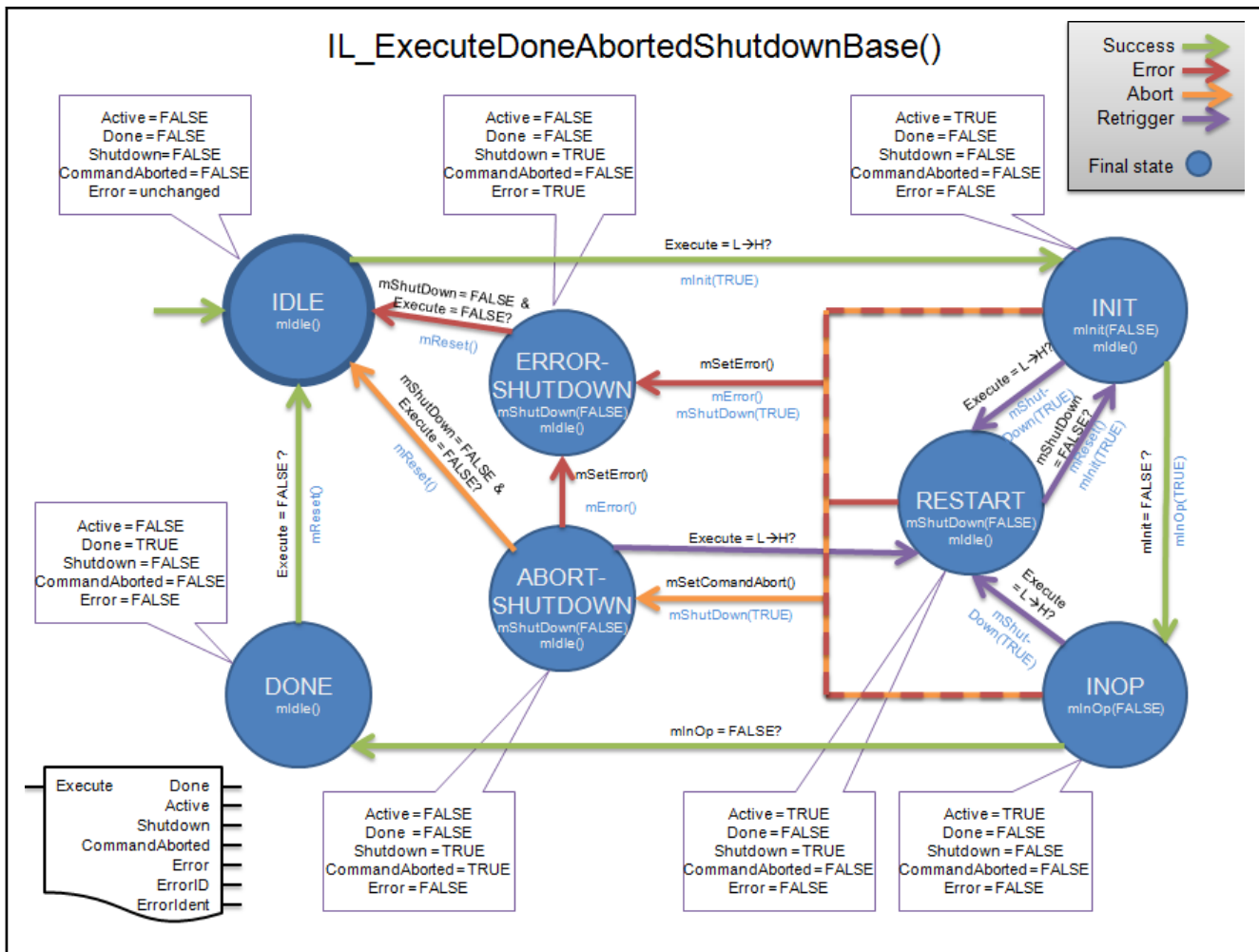


Fig. 4-38: State machine of a function block with "Execute" input and "Done", "Active", "Shutdown" and "CommandAborted" outputs.

Function blocks of this type are provided with seven internal states (see also [IL_FB_STATES](#) on page 107):

STATE_IDLE

This state represents the starting state of the function block.

It is active upon first call and it is enabled if "Execute" is set to FALSE and if the function block is in the states STATE_DONE, STATE_ABORT_SHUTDOWN or STATE_ERROR_SHUTDOWN.

When the state is entered (again), mReset() is executed. mIdle() is executed in the state.

If "Execute" is set, it is changed to STATE_INIT.

STATE_INIT

This state is used to start the function block. It runs through before the actual cyclic operation.

The "Active" output is TRUE as long as this state is active. When the state is entered, mInit(Reset:=TRUE) is called. mInit(Reset:=FALSE) and mIdle() are called in the state.

If mInit() returns FALSE, it is changed to the STATE_INOP state.

If mSetError() or mSetErrorIdent() is called, it is switched to the STATE_ERROR_SHUTDOWN state.

STATE_INOP	The cyclic task is performed here. The "Active" output is TRUE as long as this state is active. When the state is entered, mInOp(Reset:=TRUE) is called. mInOp(Reset:=FALSE) is called in the state. If mInOp() returns FALSE, the task is completed and the function block changes to the STATE_DONE state. If mSetError() or mSetErrorIdent() is called, it is switched to the STATE_ERROR_SHUTDOWN state.
STATE_DONE	State active after successful task processing. It is exited if "Execute" is FALSE, but remains active for at least one cycle. mIdle() is called in the state.
STATE_RESTART	If a positive edge at the "Execute" input is detected upon execution, the current processing is stopped and the function is started again. Therefore, the method mShutDown(TRUE) is called, when the state is entered state. The method mShutDown(FALSE) is called in the state. The outputs "Active" and "Shutdown" are enabled during execution. If the method returns mShutDown() = FALSE, it is changed to STATE_INIT.
STATE_ABORT_SHUTDOWN	This is the abort state of the function block. The outputs "Shutdown" and "CommandAborted" are active in this state. When the state is entered, mShutDown(TRUE) is called. mShutDown(FALSE) and mIdle() are called in the state. Returns the method mShutDown() = FALSE and if "Execute" is reset, it is changed to the "STATE_IDLE" state.
STATE_ERROR_SHUTDOWN	This is the error state of the function block. In this state, the outputs "Shutdown" and "Error" are enabled and the error code is displayed at "ErrorID" or "ErrorIdent". When the state is entered, mError() is called. mShutDown(TRUE) is called in the state. mShutDown(FALSE) and mIdle() are called in the state. Returns the method mShutDown() = FALSE and if "Execute" is reset, it is changed to the "STATE_IDLE" state. The "Error" output remains set.

Methods The following methods are available for the user:

Method	Function
mIdle()	Idle code as long as the function block is not in INOP
BOOL:= mInit(Reset)	Initialization code of the function block. Is processed until mInit returns FALSE. Reset at first call TRUE
BOOL:= mInOp(Reset)	Operation code of the function block. Is processed until mInOp returns FALSE. Reset at first call TRUE
BOOL:= mShutDown(Reset)	Shutdown code of the function block by abort/error/reset. Is processed until mShutDown returns FALSE. Reset at first call TRUE
mAbort()	Canceling the current task → Transition to shutdown

Method	Function
mSetError()	Called by the user when an error was detected
nSetErrorIdent()	Alternative call by the user when an error was detected

Tab. 4-26: Public methods of the IL_ExecuteDoneAbortedShutdownBase function block

Additionally, the following methods are available:

Method	Function
mError()	One-time execution when entering the ERROR state
mReset()	One-time execution when entering the states IDLE or INIT after restart
mSetOutputs()	Setting the outputs depends on the current state

Tab. 4-27: Internal methods of the IL_ExecuteDoneAbortedShutdownBase function block

Signal-time diagram

The following time response results from the state machine described above:

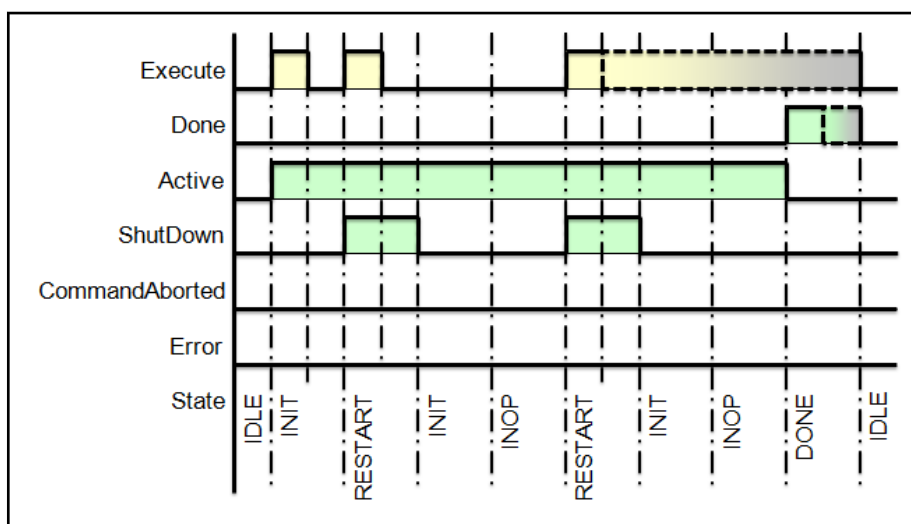


Fig. 4-39: Start and correct completion of IL_ExecuteDoneAbortedShutdownBase

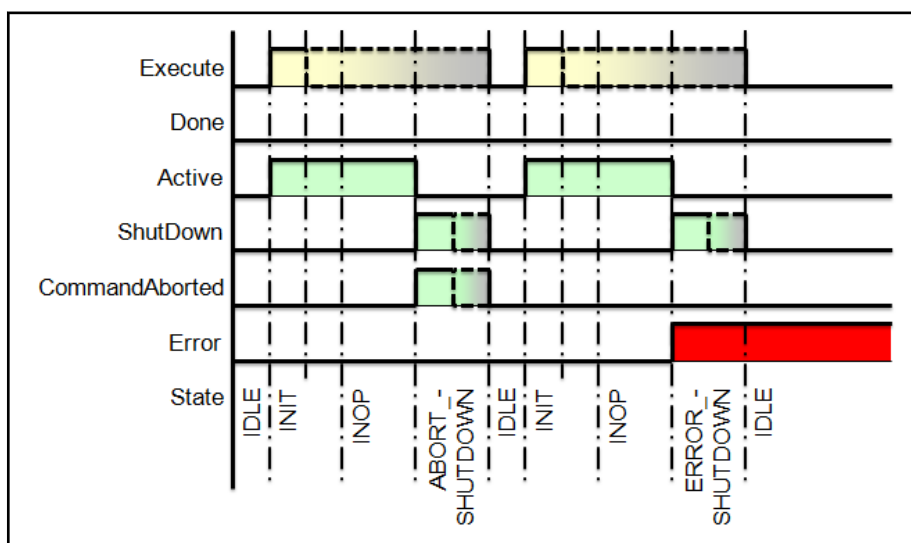


Fig. 4-40: Start and completion by abort or error of IL_ExecuteDoneAbortedShutdownBase

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00000006	16#0000FFFF	Invalid state of the state machine

Tab. 4-28: Error codes generated by IL_ExecuteDoneAbortedShutdownBase

4.3.9 IL_FB_STATES

IL_FB_STATES defines an enumeration type that contains the states of the standard state machine. The states used depend on the type of function block. STATE_IDLE is set as the starting state.

State		Description
STATE_IDLE	0	Starting state of a function block
STATE_INIT	1	Initialization of the function block State runs after the function block starts. All preparations required for the actual operations are made here. A sub-state machine is possible.
STATE_INOP	2	Working state of the function block In this state, the actual task of the function block is executed.
STATE_SHUTDOWN	3	Processing end State exists only for function blocks with a "Shutdown" output. Here, the running processes are completed in order to move the function block back to the STATE_IDLE state.
STATE_DONE	4	Processing completed State indicates that task processing was successfully completed.
STATE_ERROR	5	Error state State is active after an error occurs. It is exited by resetting "Enable" or "Execute", respectively.
STATE_COAB	6	Command canceled by another function block ("CommandAborted" output is TRUE)

Tab. 4-29: Defined states in IL_FB_STATES

4.4 Conversion (bit-stream connect)

4.4.1 Introduction and overview

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Conversion function and function blocks (bit-stream connect)	
IL_BoolToByte, page 108	8 individual bits (BOOL) are converted to one BYTE (FUN)
IL_BoolToDWord, page 109	32 individual bits (BOOL) are converted to one DWORD (FUN)
IL_BoolToWord, page 111	16 individual bits (BOOL) are converted to one WORD (FUN)
IL_ByteToDWord, page 112	4 BYTES are converted to one DWORD (FUN)
IL_ConcatByte, page 113	2 BYTES are summarized to one WORD (FUN)
IL_ConcatWord, page 114	2 WORDs are summarized to one DWORD

Tab. 4-30: RIL_Uilities, bit-stream connect

Bit-stream connect: Smaller data types are summarized to bigger ones.
An example is the function IL_BoolToByte, which converts 8 bits to one byte.

4.4.2 IL_BoolToByte

Brief description This function describes the individual bits of a byte. 8 individual bits are converted to a byte.

Interface description

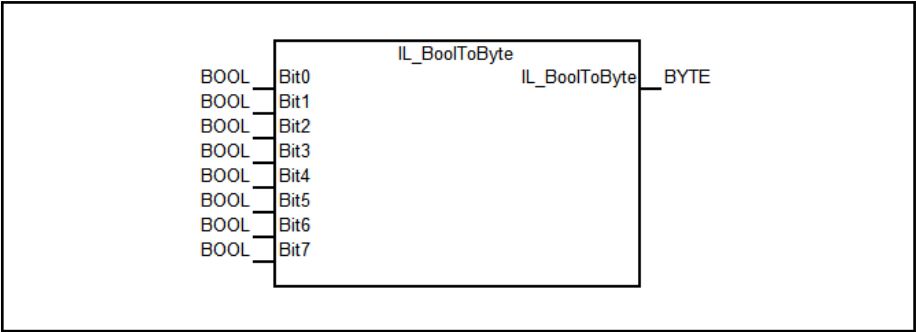


Fig. 4-41: Function IL_BoolToByte

	Name	Data type	Description
VAR_INPUT	Bit0	BOOL	Describes the position of bit 0 in the output byte
	Bit1	BOOL	Describes the position of bit 1 in the output byte
	Bit2	BOOL	Describes the position of bit 2 in the output byte
	Bit3	BOOL	Describes the position of bit 3 in the output byte
	Bit4	BOOL	Describes the position of bit 4 in the output byte
	Bit5	BOOL	Describes the position of bit 5 in the output byte
	Bit6	BOOL	Describes the position of bit 6 in the output byte
	Bit7	BOOL	Describes the position of bit 7 in the output byte
RETURN	IL_BoolToByte	BYTE	Output byte

Tab. 4-31: Interface IL_BoolToByte

Functional description This function writes the bits of the inputs to the output byte according to their significance. The byte is written in accordance with the state of all bits at the input of the function. This means that the state of the input "Bit0" is copied to bit 0 of the output byte, the state of the input "Bit1" is copied to bit 1 of the output byte, etc.

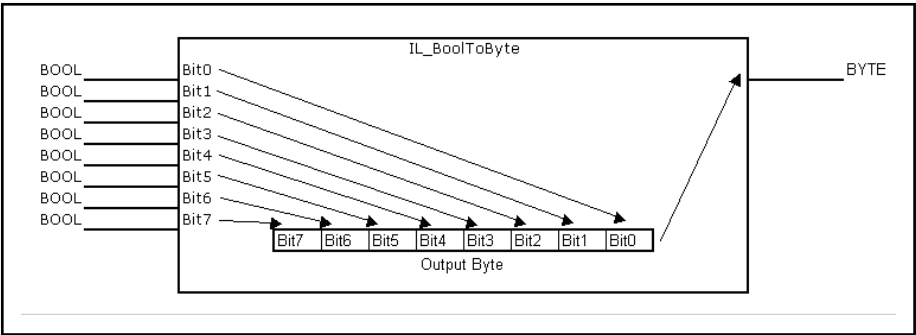


Fig. 4-42: Functional description IL_BoolToByte

Troubleshooting No troubleshooting available for this function.

4.4.3 IL_BoolToDWord

Brief description This function describes the individual bits of a DWord. 32 individual bits are converted to a double word.

Interface description

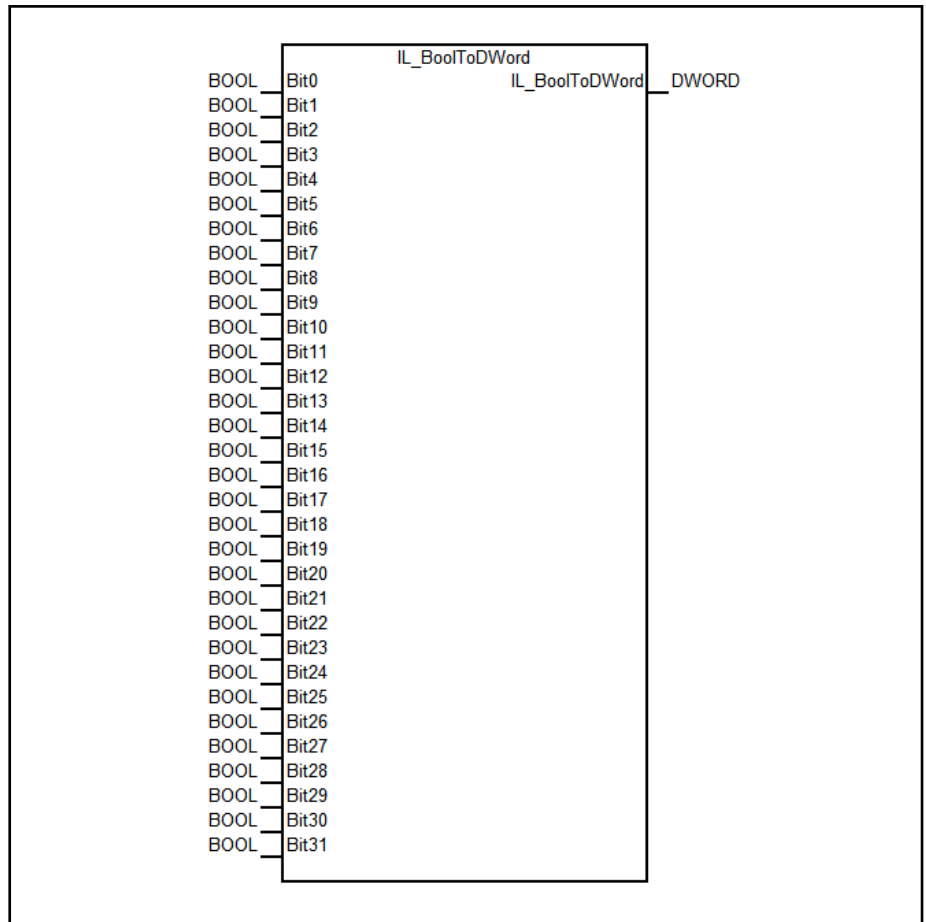


Fig. 4-43: Function IL_BoolToDWord

	Name	Data type	Description
VAR_INPUT	Bit0	BOOL	Describes the position of bit 0 in the output double word
	Bit1	BOOL	Describes the position of bit 1 in the output double word
	Bit2	BOOL	Describes the position of bit 2 in the output double word
	Bit3	BOOL	Describes the position of bit 3 in the output double word
	Bit4	BOOL	Describes the position of bit 4 in the output double word
	Bit5	BOOL	Describes the position of bit 5 in the output double word
	Bit6	BOOL	Describes the position of bit 6 in the output double word
	Bit7	BOOL	Describes the position of bit 7 in the output double word
	Bit8	BOOL	Describes the position of bit 8 in the output double word
	Bit9	BOOL	Describes the position of bit 9 in the output double word
	Bit10	BOOL	Describes the position of bit 10 in the output double word
	Bit11	BOOL	Describes the position of bit 11 in the output double word
	Bit12	BOOL	Describes the position of bit 12 in the output double word

	Name	Data type	Description
	Bit13	BOOL	Describes the position of bit 13 in the output double word
	Bit14	BOOL	Describes the position of bit 14 in the output double word
	Bit15	BOOL	Describes the position of bit 15 in the output double word
	Bit16	BOOL	Describes the position of bit 16 in the output double word
	Bit17	BOOL	Describes the position of bit 17 in the output double word
	Bit18	BOOL	Describes the position of bit 18 in the output double word
	Bit19	BOOL	Describes the position of bit 19 in the output double word
	Bit20	BOOL	Describes the position of bit 20 in the output double word
	Bit21	BOOL	Describes the position of bit 21 in the output double word
	Bit22	BOOL	Describes the position of bit 22 in the output double word
	Bit23	BOOL	Describes the position of bit 23 in the output double word
	Bit24	BOOL	Describes the position of bit 24 in the output double word
	Bit25	BOOL	Describes the position of bit 25 in the output double word
	Bit26	BOOL	Describes the position of bit 26 in the output double word
	Bit27	BOOL	Describes the position of bit 27 in the output double word
	Bit28	BOOL	Describes the position of bit 28 in the output double word
	Bit29	BOOL	Describes the position of bit 29 in the output double word
	Bit30	BOOL	Describes the position of bit 30 in the output double word
	Bit31	BOOL	Describes the position of bit 31 in the output double word
RETURN	IL_BoolToDWord	DWORD	Output double word

Tab. 4-32: Interface IL_BoolToDWord

Functional description

This function writes the bits of the inputs to the output double word according to their significance. The DWord is written in accordance with the state of all bits at the input of the function. This means that the state of the input "Bit0" is copied to bit 0 of the output double word, the state of the input "Bit1" is copied to bit 1 of the output double word, etc.

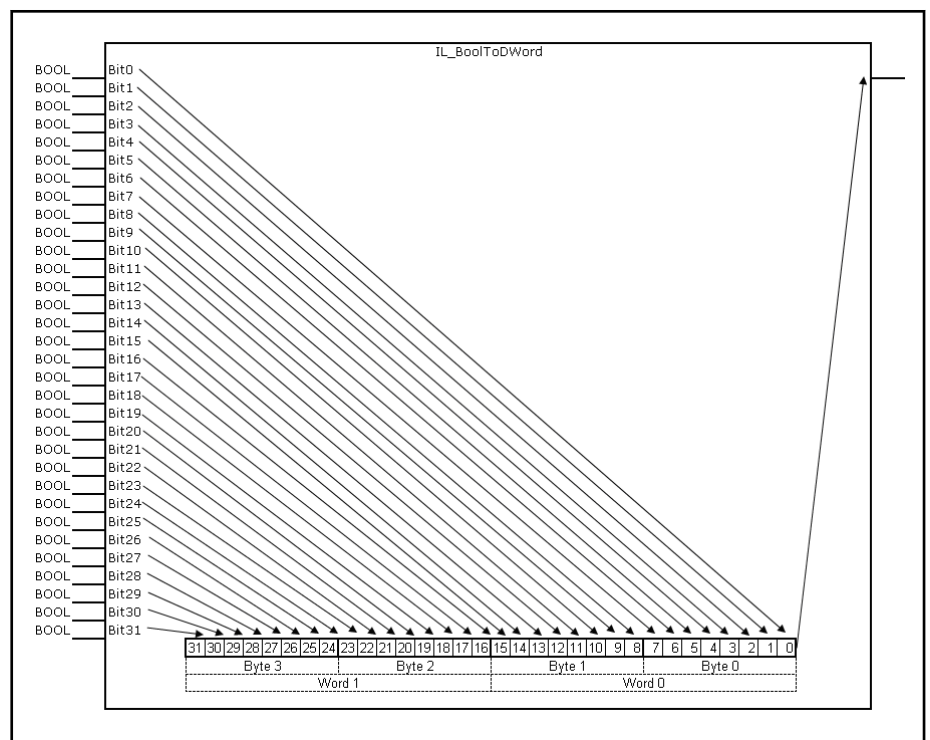


Fig. 4-44: Functional description IL_BoolToDWord

Troubleshooting

No troubleshooting available for this function.

4.4.4 IL_BoolToWord**Brief description**

This function describes the individual bits of a Word. 16 individual bits are converted to a word.

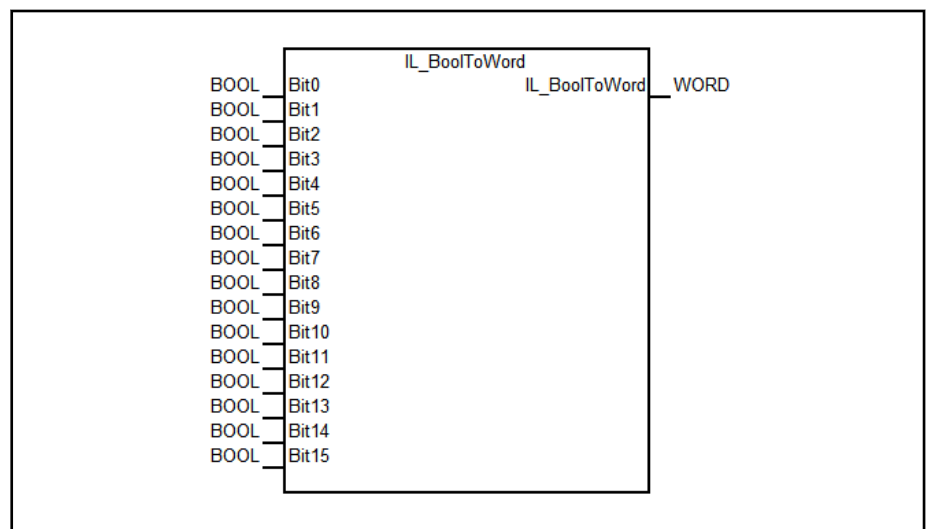
Interface description

Fig. 4-45: Function IL_BoolToWord

	Name	Data type	Description
VAR_INPUT	Bit0	BOOL	Describes the position of bit 0 in the output word
	Bit1	BOOL	Describes the position of bit 1 in the output word
	Bit2	BOOL	Describes the position of bit 2 in the output word

	Name	Data type	Description
	Bit3	BOOL	Describes the position of bit 3 in the output word
	Bit4	BOOL	Describes the position of bit 4 in the output word
	Bit5	BOOL	Describes the position of bit 5 in the output word
	Bit6	BOOL	Describes the position of bit 6 in the output word
	Bit7	BOOL	Describes the position of bit 7 in the output word
	Bit8	BOOL	Describes the position of bit 8 in the output word
	Bit9	BOOL	Describes the position of bit 9 in the output word
	Bit10	BOOL	Describes the position of bit 10 in the output word
	Bit11	BOOL	Describes the position of bit 11 in the output word
	Bit12	BOOL	Describes the position of bit 12 in the output word
	Bit13	BOOL	Describes the position of bit 13 in the output word
	Bit14	BOOL	Describes the position of bit 14 in the output word
	Bit15	BOOL	Describes the position of bit 15 in the output word
RETURN	IL_BoolToWorld	WORD	Output word

Tab. 4-33: Interface IL_BoolToWorld

Functional description

This function writes the bits of the inputs to the output word according to their significance. The word is written in accordance with the state of all bits at the input of the function. This means that the state of the input "Bit0" is copied to bit 0 of the output word, the state of the input "Bit1" is copied to bit 1 of the output word, etc.

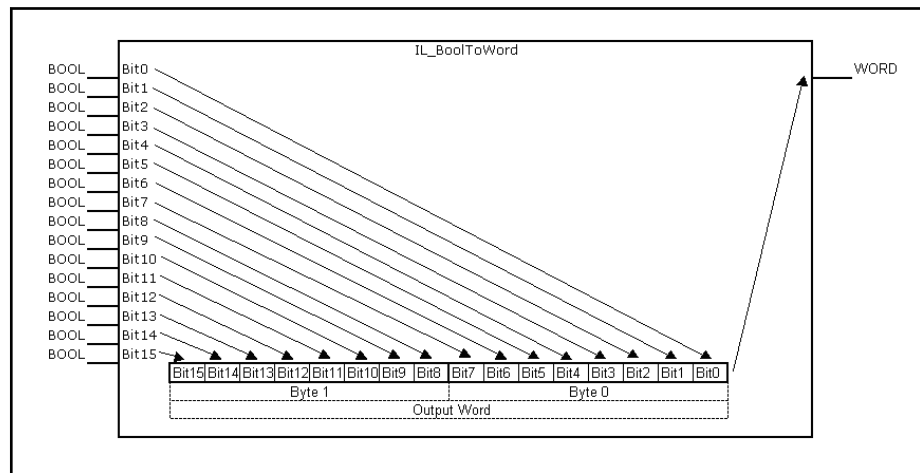


Fig. 4-46: Functional description IL_BoolToWorld

Troubleshooting

No troubleshooting available for this function.

4.4.5 IL_ByteToDWord

Brief description

This function describes the individual bytes of a DWord. 4 individual bytes are converted to a double word.

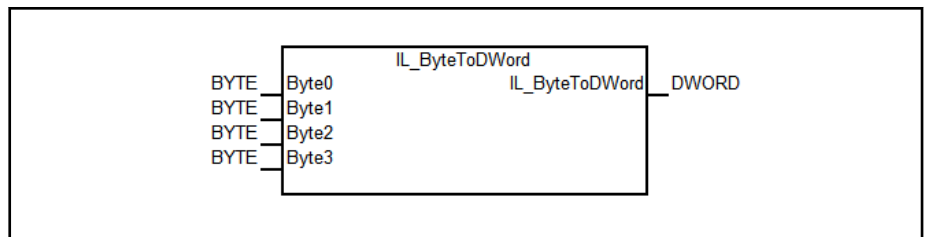
Interface description

Fig. 4-47: Function IL_ByteToDWord

	Name	Data type	Description
VAR_INPUT	Byte0	BYTE	Describes the position of byte 0 in the output double word
	Byte1	BYTE	Describes the position of byte 1 in the output double word
	Byte2	BYTE	Describes the position of byte 2 in the output double word
	Byte3	BYTE	Describes the position of byte 3 in the output double word
RETURN	IL_ByteToDWord	DWORD	Output double word

Tab. 4-34: Interface IL_ByteToDWord

Functional description

This function writes the bytes of the inputs to the output double word according to their significance. The double word is written in accordance with the state of all bytes at the input of the function. This means that the state of the input "Byte0" is copied to byte 0 of the output double word, the state of the input "Byte1" is copied to byte 1 of the output double word, etc.

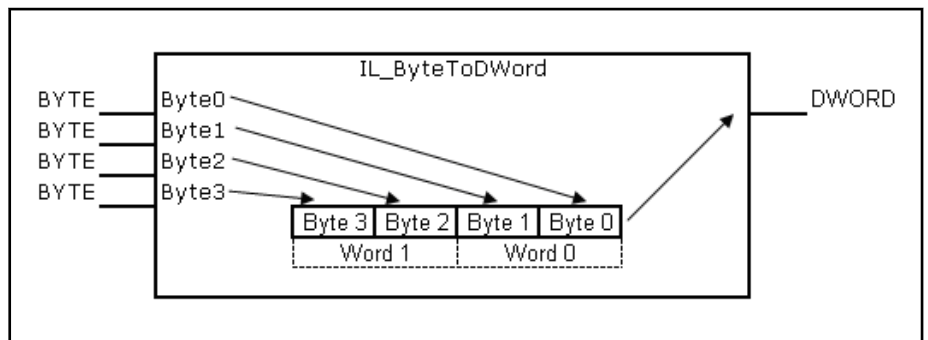


Fig. 4-48: Functional description IL_ByteToDWord

Troubleshooting

No troubleshooting available for this function.

4.4.6 IL_ConcatByte

Brief description

This function describes the individual bytes of a Word. 2 bytes are consolidated to a word.

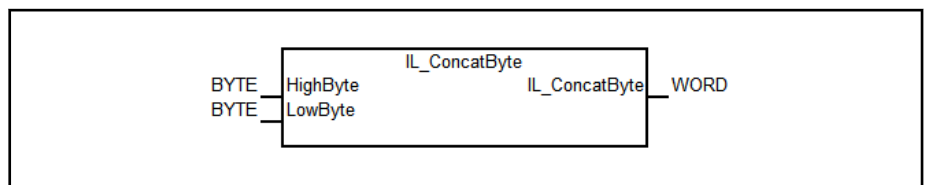
Interface description

Fig. 4-49: Function IL_ConcatByte

	Name	Data type	Description
VAR_INPUT	HighByte	BYTE	Describes the position of byte 1 in the output word
	LowByte	BYTE	Describes the position of byte 0 in the output word
RETURN	IL_ConcatByte	WORD	Output word

Tab. 4-35: Interface IL_ConcatByte

Functional description This function writes the bytes of the inputs to the output word according to their significance. The word is written in accordance with the state of all bytes at the input of the function. This means that the state of the input "HighByte" is copied to byte 1 of the output double word, the state of the input "LowByte" is copied to byte 0 of the output word, etc.

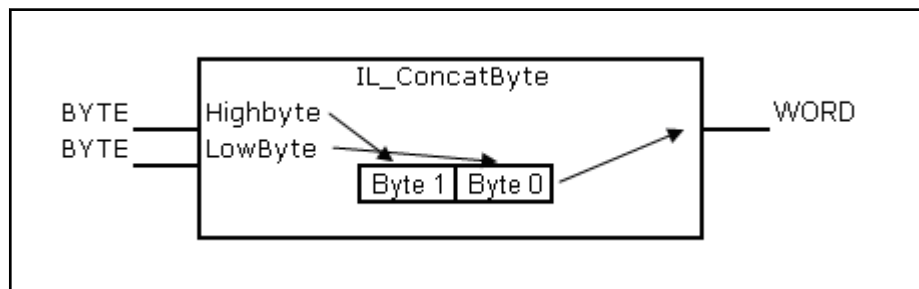


Fig. 4-50: Functional description IL_ConcatByte

Troubleshooting No troubleshooting available for this function.

4.4.7 IL_ConcatWord

Brief description This function describes the individual words of a double word. 2 words are consolidated to a double word.

Interface description

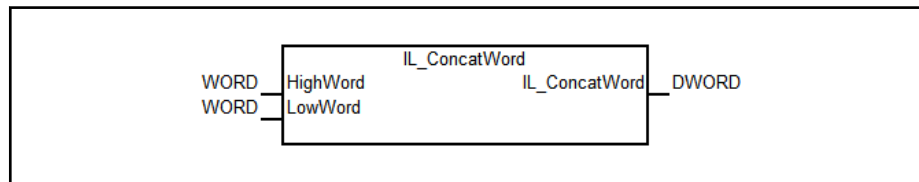


Fig. 4-51: Function IL_ConcatWord

	Name	Data type	Description
VAR_INPUT	HighWord	WORD	Describes the position of word 1 in the output double word
	LowWord	WORD	Describes the position of word 0 in the output double word
RETURN	IL_ConcatWord	DWORD	Output double word

Tab. 4-36: Interface IL_ConcatWord

Functional description This function writes the words of the inputs to the output double word according to their significance. The double word is written in accordance with the state of the words at the input of the function. This means that the state of the input "HighWord" is copied to word 1 of the output double word, the state of the input "LowWord" is copied to word 0 of the output double word, etc.

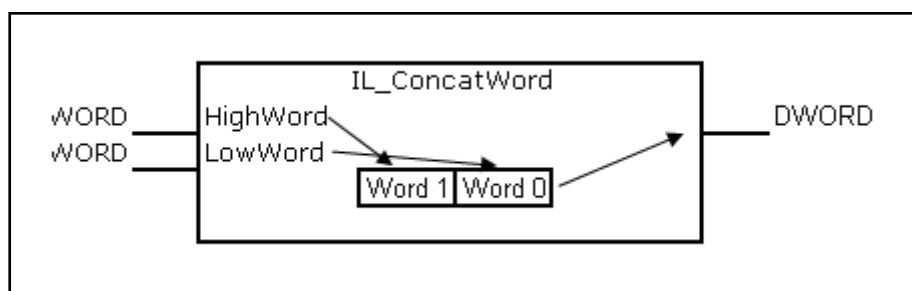


Fig. 4-52: Functional description IL_ConcatWord

Troubleshooting

No troubleshooting available for this function.

4.5 Conversion (bit-stream split)

4.5.1 Introduction and overview

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Conversion function and function blocks (bit-stream split)	
IL_ByteToBool, page 115	FB extract 8 bits (BOOL) from one BYTE
IL_DWordToBool, page 116	FB extract 32 bits (BOOL) from one DWORD
IL_DWordToByte, page 119	FB extracts 4 BYTES from one DWORD
IL_HighByte, page 119	FUN extracts a high-order BYTE from one WORD
IL_HighWord, page 120	FUN extracts a high-order WORD from one DWORD
IL_LowByte, page 120	FUN extracts a low-order BYTE from one WORD
IL_LowWord, page 121	FUN extracts a low-order WORD from one DWORD
IL_WordToBool, page 121	FB extract 16 bits (BOOL) from one WORD

Tab. 4-37: RIL_Uilities, bit-stream split

Bit-stream split: Data type is split into smaller data types.

An example is the function block IL_DWordToByte, which splits a double word in 4 bytes.

4.5.2 IL_ByteToBool

Brief description

This function block extracts individual bits from a byte. 8 individual bits are extracted from a byte.

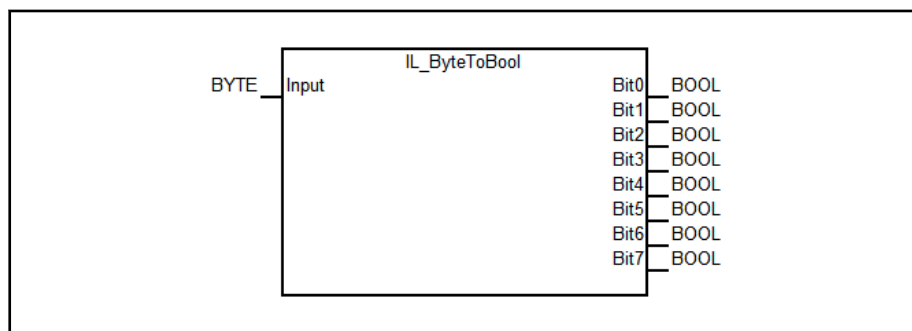
Interface description

Fig. 4-53: IL_ByteToBool function block

	Name	Data type	Description
VAR_INPUT	Input	BYTE	Input byte from which the bits are extracted
VAR_OUPUT	Bit0	BOOL	Bit 0 of the input byte
	Bit1	BOOL	Bit 1 of the input byte
	Bit2	BOOL	Bit 2 of the input byte
	Bit3	BOOL	Bit 3 of the input byte
	Bit4	BOOL	Bit 4 of the input byte
	Bit5	BOOL	Bit 5 of the input byte
	Bit6	BOOL	Bit 6 of the input byte
	Bit7	BOOL	Bit 7 of the input byte

Tab. 4-38: Interface IL_BoolToBool

Functional description

With this function block, the individual bits of a byte are determined. The state of the input "Input" is registered and the status of the individual bits is copied to the output variables "Bit0" .. "Bit7".

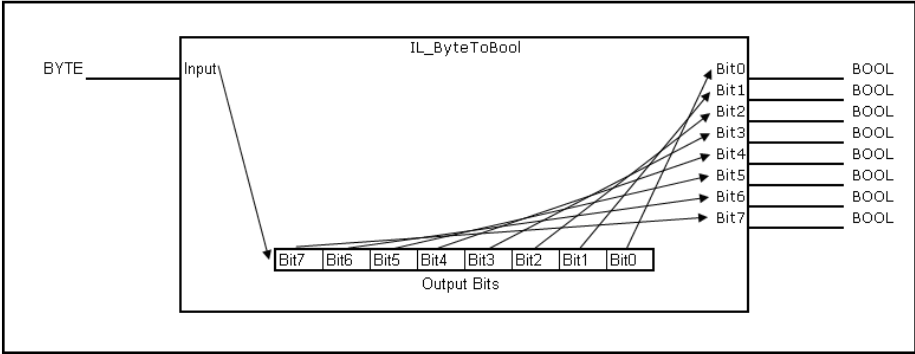


Fig. 4-54: Functional description IL_ByteToBool

Troubleshooting

No error handling is available for this function block.

4.5.3

IL_DWordToBool

Brief description

This function block extracts individual bits from a double word. 32 individual bits are extracted from a DWord.

Interface description

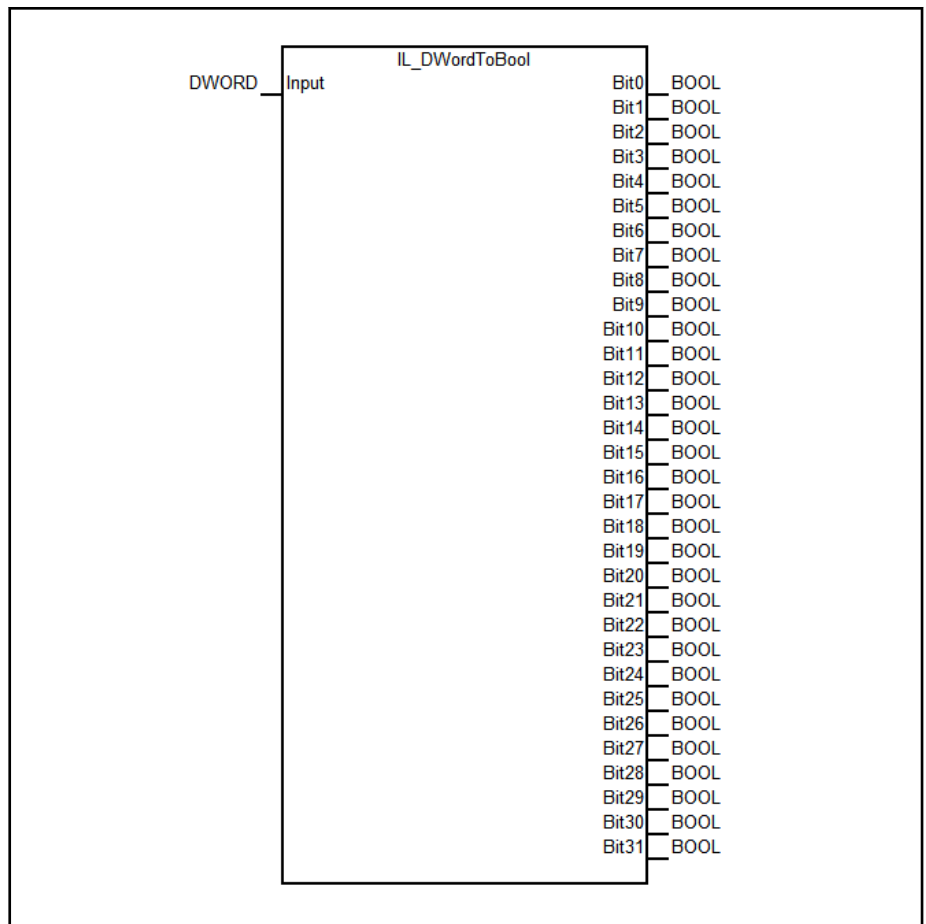


Fig. 4-55: IL_DWordToBool function block

	Name	Data type	Description
VAR_INPUT	Input	DWORD	Input double word from which the bits are extracted
VAR_OUPUT	Bit0	BOOL	Bit 0 of the input double word
	Bit1	BOOL	Bit 1 of the input double word
	Bit2	BOOL	Bit 2 of the input double word
	Bit3	BOOL	Bit 3 of the input double word
	Bit4	BOOL	Bit 4 of the input double word
	Bit5	BOOL	Bit 5 of the input double word
	Bit6	BOOL	Bit 6 of the input double word
	Bit7	BOOL	Bit 7 of the input double word
	Bit8	BOOL	Bit 8 of the input double word
	Bit9	BOOL	Bit 9 of the input double word
	Bit10	BOOL	Bit 10 of the input double word
	Bit11	BOOL	Bit 11 of the input double word
	Bit12	BOOL	Bit 12 of the input double word
	Bit13	BOOL	Bit 13 of the input double word

	Name	Data type	Description
	Bit14	BOOL	Bit 14 of the input double word
	Bit15	BOOL	Bit 15 of the input double word
	Bit16	BOOL	Bit 16 of the input double word
	Bit17	BOOL	Bit 17 of the input double word
	Bit18	BOOL	Bit 18 of the input double word
	Bit19	BOOL	Bit 19 of the input double word
	Bit20	BOOL	Bit 20 of the input double word
	Bit21	BOOL	Bit 21 of the input double word
	Bit22	BOOL	Bit 22 of the input double word
	Bit23	BOOL	Bit 23 of the input double word
	Bit24	BOOL	Bit 24 of the input double word
	Bit25	BOOL	Bit 25 of the input double word
	Bit26	BOOL	Bit 26 of the input double word
	Bit27	BOOL	Bit 27 of the input double word
	Bit28	BOOL	Bit 28 of the input double word
	Bit29	BOOL	Bit 29 of the input double word
	Bit30	BOOL	Bit 30 of the input double word
	Bit31	BOOL	Bit 31 of the input double word

Tab. 4-39: Interface IL_DWordToBool

Functional description

With this function block, the individual bits of a double word are determined. The state of the input "Input" is registered and the status of the individual bits is copied to the output variables "Bit0" .. "Bit31".

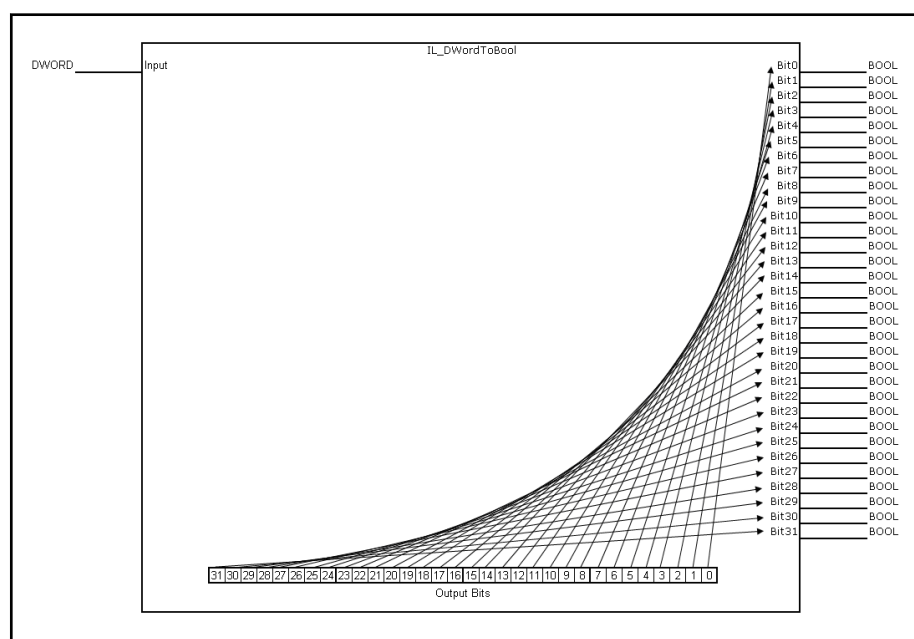


Fig. 4-56: Functional description IL_DWordToBool

Troubleshooting

No error handling is available for this function block.

4.5.4 IL_DWordToByte

Brief description This function block extracts individual bytes from a DWord. 4 individual bytes are extracted from a double word.

Interface description

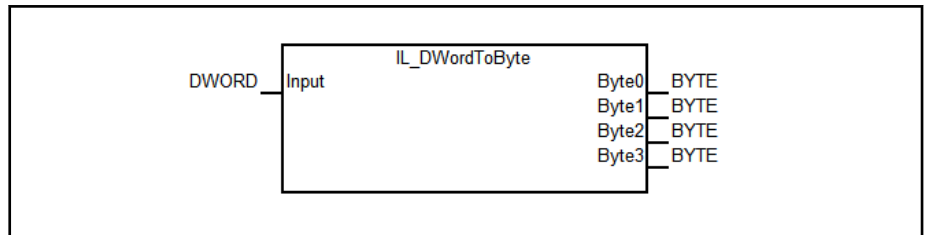


Fig. 4-57: IL_DWordToByte function block

	Name	Data type	Description
VAR_INPUT	Input	DWORD	Input double word from which the bytes are extracted
VAR_OUTPUT	Byte0	BYTE	Byte 0 of the input double word
	Byte1	BYTE	Byte 1 of the input double word
	Byte2	BYTE	Byte 2 of the input double word
	Byte3	BYTE	Byte 3 of the input double word

Tab. 4-40: Interface IL_DWordToByte

Functional description

With this function block, the individual bytes of a double word are determined. The state of the input "Input" is registered and the status of the individual bytes is copied to the output variables "Byte0" .. "Byte3".

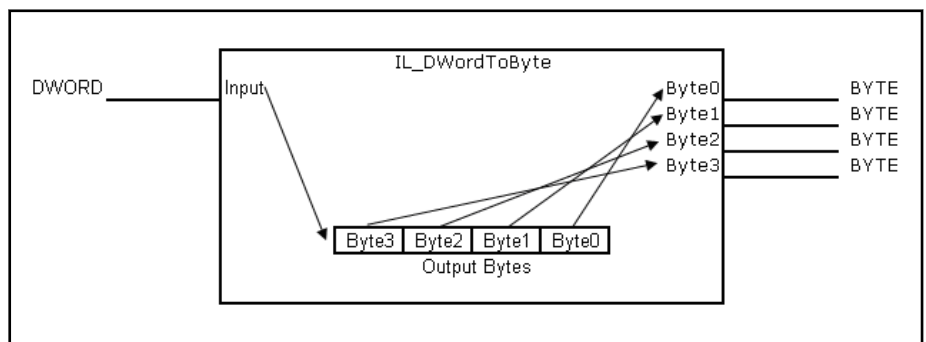


Fig. 4-58: Functional description IL_DWordToByte

Troubleshooting

No error handling is available for this function block.

4.5.5 IL_HighByte

Brief description This function extracts the high-order byte from a word.

Interface description

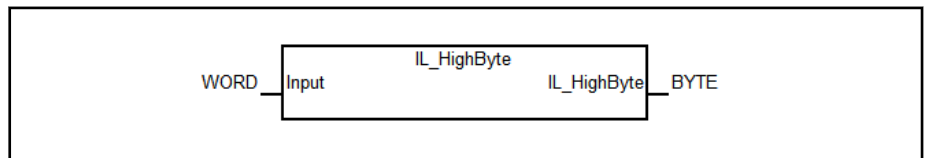


Fig. 4-59: Function IL_HighByte

	Name	Data type	Description
VAR_INPUT	Input	WORD	Input word from which the byte is extracted
RETURN	IL_HighByte	BYTE	High-order byte of the input word

Tab. 4-41: Interface IL_HighByte

Functional description

With this function, the high-order byte of a word is determined. The state of the input "Input" is registered and the status of the high-order byte is copied to the function output.

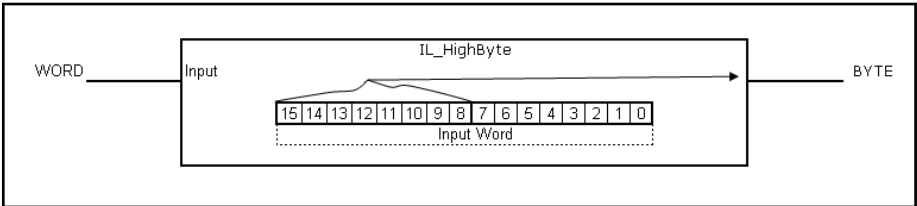


Fig. 4-60: Functional description IL_HighByte

Troubleshooting

No troubleshooting available for this function.

4.5.6

IL_HighWord

Brief description

This function extracts the high-order word from a double word.

Interface description

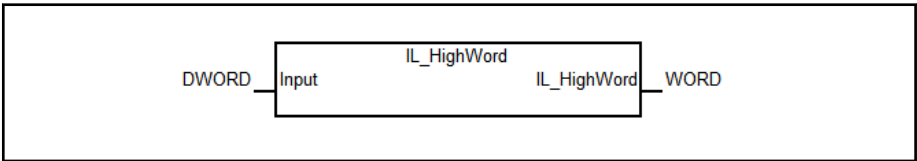


Fig. 4-61: Function IL_HighWord

	Name	Data type	Description
VAR_INPUT	Input	DWORD	Input double word from which the word is extracted
RETURN	IL_HighWord	WORD	High-order word of the input double word

Tab. 4-42: Interface IL_HighWord

Functional description

With this function, the high-order word of a double word is determined. The state of the input "Input" is registered and the status of the high-order word is copied to the function output.

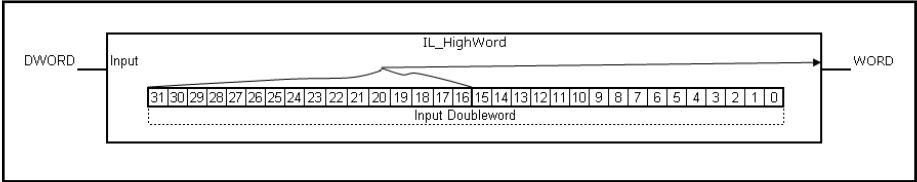


Fig. 4-62: Functional description IL_HighWord

Troubleshooting

No troubleshooting available for this function.

4.5.7

IL_LowByte

Brief description

This function extracts the low-order byte from a word.

Interface description

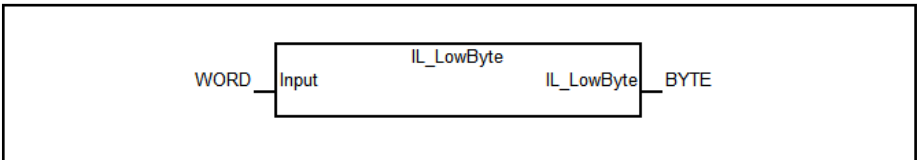


Fig. 4-63: Function IL_LowByte

	Name	Data type	Description
VAR_INPUT	Input	WORD	Input word from which the byte is extracted
RETURN	IL_LowByte	BYTE	Lw-order byte of the input word

Tab. 4-43: Interface IL_LowByte

Functional description

With this function, the low-order byte of a word is determined. The state of the input "Input" is registered and the status of the low-order byte is copied to the function output.

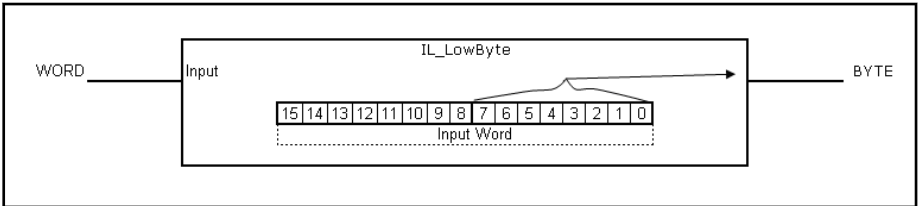


Fig. 4-64: Functional description IL_LowByte

Troubleshooting

No troubleshooting available for this function.

4.5.8

IL_LowWord

Brief description

This function extracts the low-order word from a double word.

Interface description

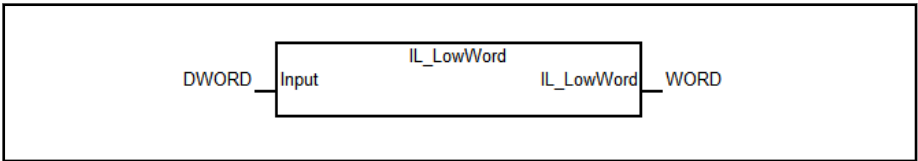


Fig. 4-65: Function IL_LowWord

	Name	Data type	Description
VAR_INPUT	Input	DWORD	Input double word from which the word is extracted
RETURN	IL_LowWord	WORD	Low-order word of the input double word

Tab. 4-44: Interface IL_LowWord

Functional description

With this function, the low-order word of a double word is determined. The state of the input "Input" is registered and the status of the low-order word is copied to the function output.

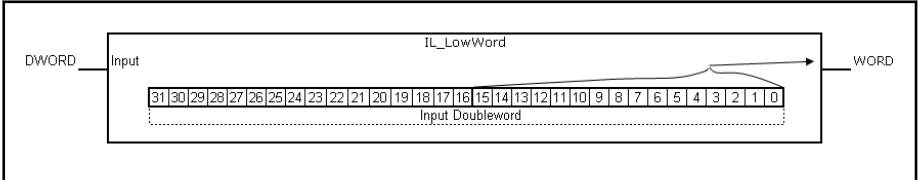


Fig. 4-66: Functional description IL_LowWord

Troubleshooting

No troubleshooting available for this function.

4.5.9

IL_WordToBool

Brief description

This function block extracts individual bits from a word. 16 individual bits are extracted from a word.

Interface description

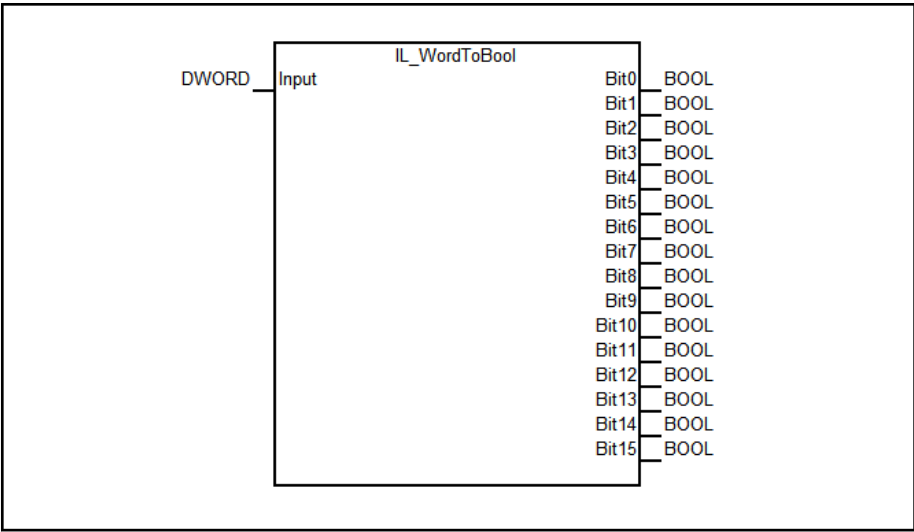


Fig. 4-67: IL_WordToBool function block

	Name	Data type	Description
VAR_INPUT	Input	WORD	Input word from which the bits are extracted
VAR_OUPUT	Bit0	BOOL	Bit 0 of the input word
	Bit1	BOOL	Bit 1 of the input word
	Bit2	BOOL	Bit 2 of the input word
	Bit3	BOOL	Bit 3 of the input word
	Bit4	BOOL	Bit 4 of the input word
	Bit5	BOOL	Bit 5 of the input word
	Bit6	BOOL	Bit 6 of the input word
	Bit7	BOOL	Bit 7 of the input word
	Bit8	BOOL	Bit 8 of the input word
	Bit9	BOOL	Bit 9 of the input word
	Bit10	BOOL	Bit 10 of the input word
	Bit11	BOOL	Bit 11 of the input word
	Bit12	BOOL	Bit 12 of the input word
	Bit13	BOOL	Bit 13 of the input word
	Bit14	BOOL	Bit 14 of the input word
	Bit15	BOOL	Bit 15 of the input word

Tab. 4-45: Interface IL_WordToBool

Functional description

With this function block, the individual bits of a word are determined. The state of the input "Input" is registered and the status of the individual bits is copied to the output variables "Bit0" .. "Bit15".

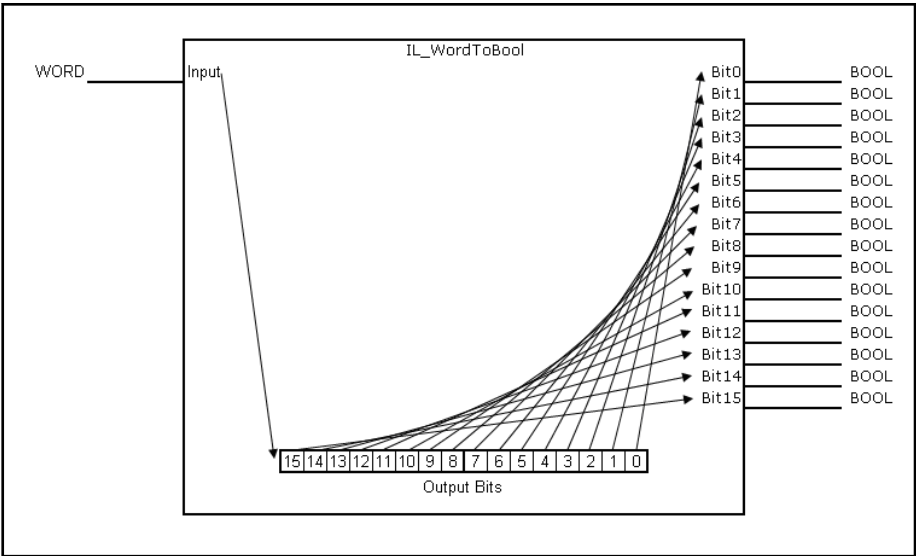


Fig. 4-68: Functional description IL_WordToBool

Troubleshooting

No error handling is available for this function block.

4.6

Conversion (ByteSwap)

4.6.1

Introduction and overview

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Conversion functions and function blocks (ByteSwap)	
IL_SwapDWord, page 123	FUN exchanges BYTEs of a DWORDs (3 2 1 0 -> 0 1 2 3) Note: Not PLC-compatible!
IL_SwapWord, page 124	FUN exchanges BYTEs of a WORD (1 0 -> 0 1)

Tab. 4-46: RIL_Uilities, ByteSwap

Byte swap: The lower and the upper half of the data type are swapped.
An example is the function IL_SwapWord, which swaps the low-order byte of the input word with the high-order one.

4.6.2

IL_SwapDWord

Brief description

This function swaps the bytes of a double word 3 2 1 0 --> 0 1 2 3.

Interface description

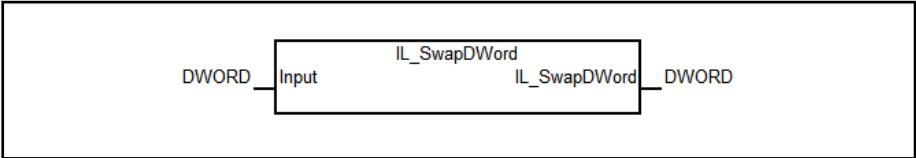


Fig. 4-69: Function IL_SwapDWord

	Name	Data type	Description
VAR_INPUT	Input	DWORD	Input double word
RETURN	IL_Swap-Word	DWORD	Output double word with swapped bytes

Tab. 4-47: Interface IL_SwapDWord

Functional description This function swaps the bytes of a double word in their sequence.
The state of the "Input" input is detected.
The byte sequence changes from 3 2 1 0 to 0 1 2 3.
The result is copied to the function output.

 The function name is not according to IEC 61131-3.

Troubleshooting No troubleshooting available for this function.

4.6.3 IL_SwapWord

Brief description This function swaps the high-order and the low-order byte of a word.

Interface description

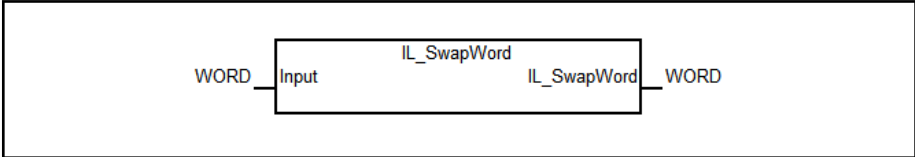


Fig. 4-70: Function IL_SwapWord

	Name	Data type	Description
VAR_INPUT	Input	WORD	Input word
RETURN	IL_SwapWord	WORD	Output word with which the high-order byte was swapped with the low-order byte.

Tab. 4-48: Interface IL_SwapWord

Functional description With this function, the high-order byte of a word is swapped with the low-order byte. The state of the input "Input" is registered and the high-order and the low-order byte are swapped and copied to the function output.

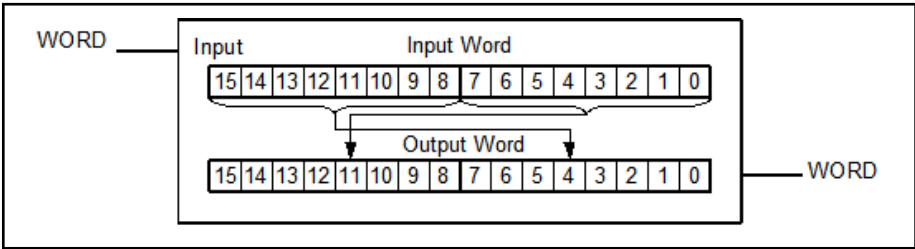


Fig. 4-71: Functional description IL_SwapWord

Troubleshooting No troubleshooting available for this function.

4.7 Checksums

4.7.1 Introduction and overview

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description

Checksums (CRC16)	
IL_CRC16, page 125	FB computes CRC16 checksum using a date

Tab. 4-49: *RIL_Uilities, CRC16*

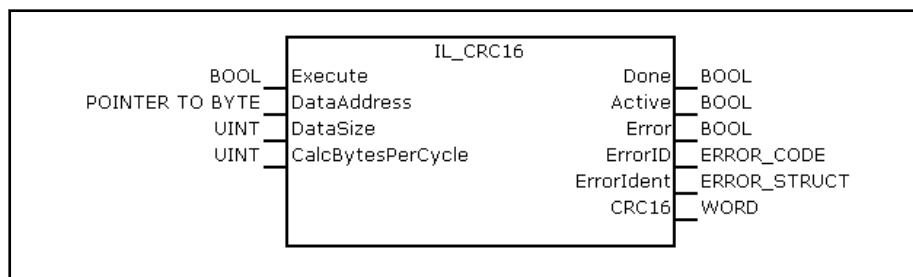
This sections describes functions and function blocks calculating checksums. Currently, there is only function block implemented into this function group.

4.7.2 IL_CRC16

Brief description

The function block IL_CRC16 calculates a CRC16 checksum via a date.

Interface description

Fig. 4-72: *Function block IL_CRC16*

	Name	Type	Comment
VAR_INPUT	Execute	BOOL	Processing enabled (cyclic, edge-controlled)
	DataAddress	POINTER TO BYTE	Address of the date to be used for calculating the checksum
	DataSize	UINT	Size of the date in bytes
	CalcBytesPerCycle	UINT	Size in bytes of the partial data block to be calculated per call
VAR_OUTPUT	Done	BOOL	Processing completed by the function block
	Active	BOOL	The function block is active
	Error	BOOL	Indicates that an error occurred
	ErrorID	ERROR_CODE	Error cause
	ErrorIdent	ERROR_STRUCT	Detailed error cause Error codes, page 125
	CRC16	WORD	Calculated checksum

Tab. 4-50: *Function block IL_CRC16*

Functional description

When there is a rising edge at the Enable input, the input values are applied and calculation of the checksum is started.

The values of the address ("DataAddress") and its size ("DataSize") of the date must be valid values.

If input "CalcBytesPerCycle" is 0, the checksum is completely calculated upon the first call. Else, only the number of bytes specified in "CalcBytesPerCycle" is calculated before the function block returns. This prevents the processing load from becoming too high.

The value at output "CRC16" is only valid if output "Done" is set.

Error codes

The function block uses error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000002	16#00000001	The "DataAddress" input is 0.
INPUT_RANGE_ERROR	16#00000002	16#00000002	The "DataSize" input is 0.

Tab. 4-51: Error codes generated by IL_CRC16

4.8 Conversion (data type change)

4.8.1 Introduction and overview

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Conversion (data type "Change")	
IL_AsciiToByte on page 126	FUN converts any ASCII character to a byte code
IL_ByteToAscii on page 126	FUN converts any byte code to an ASCII character
IL_RealToString on page 127	FUN converts REAL value to a STRING
IL_LRealToString on page 127	FUN converts (L)REAL value format-controlled to a STRING

Tab. 4-52: RIL_Uilities, data type change

The functions interpret their input data according to the guideline given in the description.

4.8.2 IL_AsciiToByte

Brief description This function converts an arbitrary ASCII sign to the corresponding byte code.

Interface description

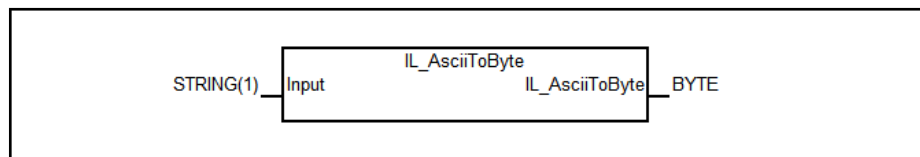


Fig. 4-73: Function IL_AsciiToByte

	Name	Data type	Description
VAR_INPUT	Input	BYTE	Input byte containing the byte code
RETURN	IL_AsciiToByte	STRING[1]	The function output contains the ASCII sign which corresponds to the byte code in the input string.

Tab. 4-53: Interface IL_AsciiToByte

Functional description With this function, the ASCII sign of the byte code in the input byte is determined. The state of the input "Input" is registered and the corresponding ASCII sign is copied to the function output.

Troubleshooting No troubleshooting available for this function.

4.8.3 IL_ByteToAscii

Brief description This function converts an arbitrary byte code to the corresponding ASCII sign.

Interface description

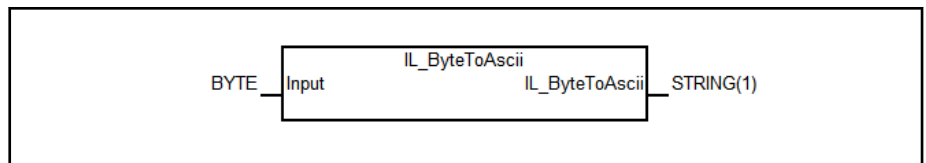


Fig. 4-74: Function IL_ByteToAscii

	Name	Data type	Description
VAR_INPUT	Input	BYTE	Input byte containing the byte code to be converted
RETURN	IL_ByteToAscii	STRING[1]	Function output contains the ASCII sign of the byte code in the input string

Tab. 4-54: Interface IL_ByteToAscii

Functional description With this function, the byte code of the ASCII sign in the input string is determined. The state of the input "Input" is registered and the corresponding ASCII value of the sign is copied to the function output.

Troubleshooting No troubleshooting available for this function.

4.8.4 IL_RealToString

Brief description This function converts a REAL value to a string.

Interface description

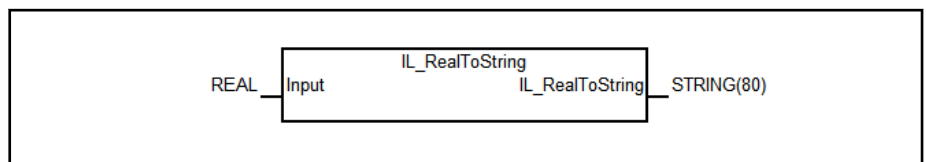


Fig. 4-75: Function IL_RealToString

	Name	Data type	Description
VAR_INPUT	Input	REAL	Input byte containing the figure to be converted
RETURN	IL_RealTo-String	STRING	Function output contains the real value as string.

Tab. 4-55: Interface IL_RealToString

Functional description With this function, the content of the input "Input" is registered, converted to a string and copied to the output of the function. As of values lower than "0.1", IndraLogic automatically uses the exponential notation (e.g. 0.0126 is written as 1.26e-002). The function "IL_RealToString" ignores this and displays the real value as string in the decimal notation. The function "IL_RealToString" only considers values up to the fourth decimal place (e.g. the real 0.0123789 becomes string 0.0124)!

Troubleshooting No troubleshooting available for this function.

4.8.5 IL_LRealToString

Brief description This function converts an (L)REAL value to a STRING. The number is converted with regard to the content of the format string.



If a REAL number is applied to an LREAL input, it is implicitly converted from REAL into LREAL.

Interface description

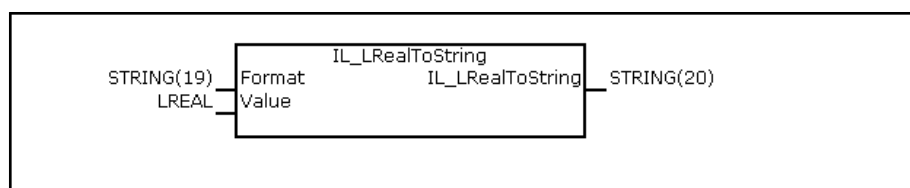


Fig. 4-76: IL_LRealToString function

	Name	Data type	Description
VAR_INPUT	Format	STRING(19)	Format string to represent "Value"
	Value	LREAL	Number to be converted
Return value	IL_LRealTo-String	STRING(20)	Function output contains LREAL value as STRING

Tab. 4-56: IL_LRealToString interface

Functional description

This function converts an LREAL value to a STRING. The number is formatted with regard to the content of the format string.

The format string determines the number of decimal places in front of and after the comma. It can output leading zeroes or selected between decimal point and comma.

Syntax of the format string:

[0|#] [{#}] [.,] [{#}]

'#': Places(s) to be output in from of and after the comma

'0': Output of leading zeroes

':': Decimal separator (US)

',' : Decimal separator (DE)

Effect of the format string

LREAL number	Format	Result	Notes
-1.23456789E02	###.###	-123.457	The last place is rounded
1.23456789E02	###.###	_123.457	Positive numbers are always provided with a leading space
1.23456789E02	###,###	_123,457	Uses a comma as decimal separator
1.2	###.###	__1.200	Format is always filled left and right
-1.2345678E-06	###.###	__-0.000	if a negative value is rounded to zero, "-0" is displayed
1.23456789E05	###.###	_12345.679	if the number cannot be output in the format, the format is extended
-1.23456789E02	#####	__-123	Output as whole number
1.23456789E02	0#####	_00123	Output with leading zeroes
-1.23456789E02	0#####	-00123	As above

LREAL number	Format	Result	Notes
1.2345E02	(blank)	1.2345E02	An empty format is converted using the standard function LREAL_TO_STRING()
1.23456789E02	##0.##	Format error	If the format string is faulty, "Format error" is output

Remarks

Values allowed for LREAL:

$\pm(2.2250738585072014e-308 \dots 1.7976931348623158e+308)$

Values $> 9.0007199254741e+15$ or $< -9.0007199254741e+15$ ($\pm(2e+53 - 1)$) are only converted using LREAL_TO_STRING().

LREAL	Format	STRING
-2.2250738585072014e-308	'###.###'	__-0.000
-999999.999999	'0#####'	-0001000000
0.1	'###.###'	__0.100
0.0314e+4	'####.###'	__314.000
3.402823466e+38	"	3.402823466e+38

Tab. 4-57: Conversion examples

Troubleshooting

If a value cannot be output in the specified format, the format is extended accordingly.

If an incorrect format is specified, "Format Error" is output.

4.9 Debugging

4.9.1 Introduction and overview

Functions to process high-resolution CPU ticks are available. These functions can be used to determine the operation time of program modules:

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Debugging	
IL_HighResTimeDiff, page 129	Calculates the time difference of two high-resolution time ticks of the system in microseconds
IL_HighResTimeTick, page 130	Reads the high-resolution time cycle of the system

Tab. 4-58: RIL_Uilities, Debugging

4.9.2 IL_HighResTimeDiff

Brief description

The function IL_HighResTimeDiff calculates the time difference of two high-resolution time ticks of the system in microseconds.

Interface description

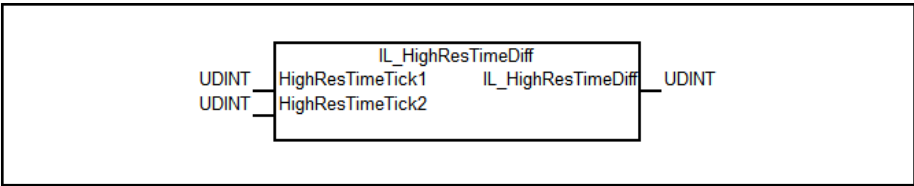


Fig. 4-77: Function IL_HighResTimeDiff

	Name	Data type	Description
VAR_INPUT	HighResTimeTick1	UDINT	High-resolution time tick of the system before the code section
	HighResTimeTick2	UDINT	High-resolution time tick of the system after the code section
RETURN	IL_HighResTimeDiff	UDINT	Time difference of two high-resolution time ticks of the system in µs

Tab. 4-59: Interface IL_HighResTimeDiff

Specification The function IL_HighResTimeDiff is used for the determination of the operation time of a code section together with the function IL_HighResTimeTick

 Depending on the system, the reading of a high-resolution time tick takes ca. one to two microseconds which can be additionally taken into consideration for the determination of the time difference by the user, if necessary.

Functional description The function IL_HighResTimeDiff calculates the time difference of two high-resolution time ticks of the system in microseconds.

Troubleshooting No troubleshooting available for this function.

Use case also refer to [IL_HighResTimeTick](#), page 131.

4.9.3 IL_HighResTimeTick

Brief description The function IL_HighResTimeTick reads the high-resolution time tick of the system.

Interface description

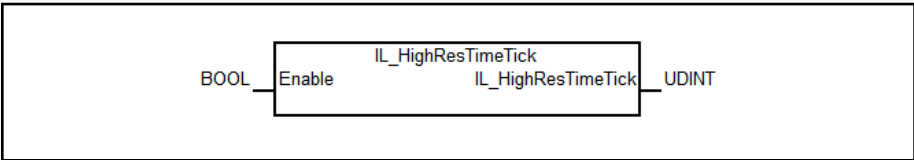


Fig. 4-78: Function IL_HighResTimeTick

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
RETURN	IL_HighResTimeTick	UDINT	High-resolution time tick of the system

Tab. 4-60: Interface IL_HighResTimeTick

Specification The function IL_HighResTimeTick is used for the determination of the operation time of a code section together with the function IL_HighResTimeDiff

 The return value of the function IL_HighResTimeTick should not be directly used as a time value due to the special system-dependent time base.

Functional description The function IL_HighResTimeTick cyclically reads the high-resolution time tick of the system after processing enabled for the function with "Enable".

Troubleshooting No troubleshooting available for this function.

Use case In this program example, the operation time of a program block is measured.

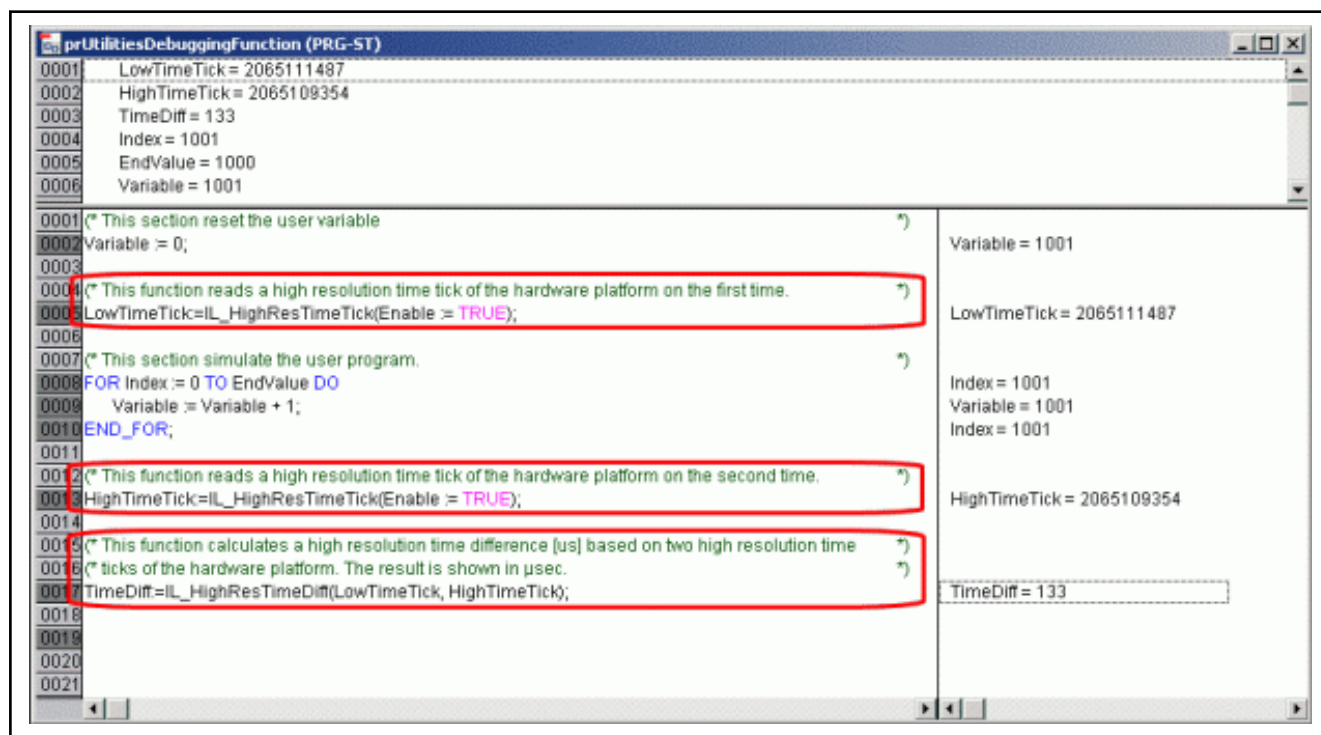


Fig. 4-79: Use case of IL_HighResTimeTick

4.10 FlipFlops

4.10.1 Introduction and overview

This function block extends the trigger function blocks provided by the default library.

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
FlipFlops	
IL_Toggle on page 131	Settable and resettable FlipFlop

Tab. 4-61: RIL_Uilities, FlipFlops

4.10.2 IL_Toggle

Brief description

This function block changes the status of its Boolean output with every rising edge at the "Execute" input. Two additional inputs allow for setting the Boolean output to TRUE or FALSE.

Interface description

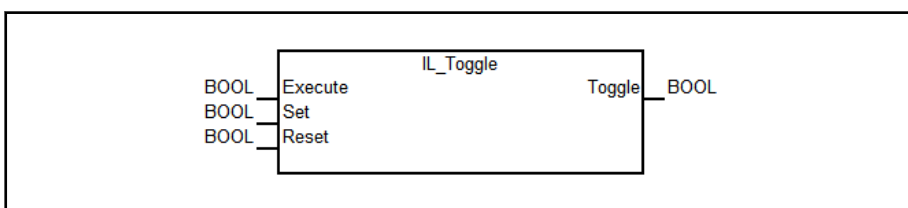


Fig. 4-80: IL_Toggle function block

	Name	Data type	Description
VAR_INPUT	Execute	BOOL	Rising edge changes the status of the output "Toggle"
	Set	BOOL	Sets the output Toggle to TRUE
	Reset	BOOL	Sets the output Toggle to FALSE
VAR_OUPUT	Toggle	BOOL	Current state

Tab. 4-62: Interface IL_Toggle

Signal-time diagram

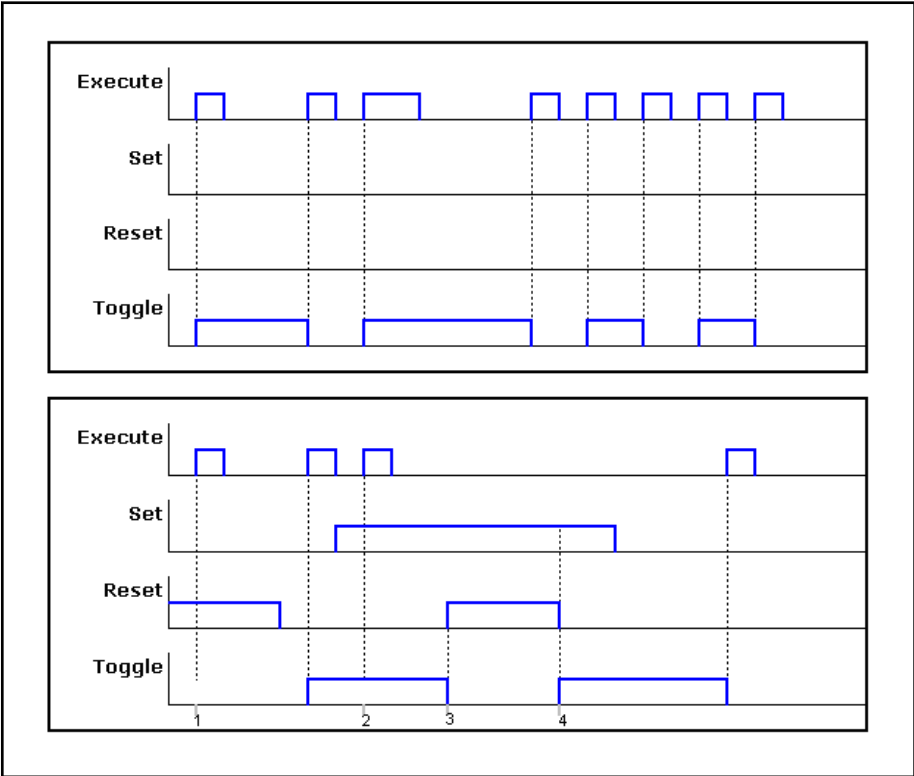


Fig. 4-81: Signal-time diagram of the IL_Toggle function block

- Top: Behavior of "Toggle" when "Set" and "Reset" remain FALSE
Down: Behavior of "Toggle" when "Set", "Reset" and "Execute" interact
1. "Toggle" remains FALSE since "Reset" is TRUE
 2. "Toggle" remains TRUE since "Set" is TRUE
 3. "Toggle" becomes FALSE since "Reset" dominates
 4. "Toggle" becomes TRUE since "Set" is TRUE

Functional description

The function block IL_Toggle changes the status of its output "Toggle" between TRUE and FALSE with every rising edge at the input "Execute". The input "Set" fixes the output "Toggle" to TRUE. The input "Reset" fixes the output "Toggle" to FALSE. If both inputs "Set" and "Reset" are TRUE, "Reset" dominates so that the output "Toggle" is set to FALSE. As long as "Set" or "Reset" are TRUE, signal changes at "Execute" do not affect the output "Toggle".

Troubleshooting

No error handling is available for this function block.

4.11 Mathematical functions

4.11.1 Introduction and overview

The following function blocks provide mathematical functions.



In 14VRS, the mentioned function blocks changed ML_* to IL_* in their names and moved from "ML_TechBase" to "RIL_Uilities".

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Mathematical functions	
IL_InterpolationLinear, page 133,	Calculates intermediate points and linearizations for example. The function block uses the structure IL_COORD_PAIR_TYPE01, page 136
IL_MaxValue, page 135,	Indicates the maximum value of all occurred values

Tab. 4-63: RIL_Uilities, mathematical functions

4.11.2 IL_InterpolationLinear

Brief description The function block IL_InterpolationLinear cyclically calculates the y-value associated to a given x-value by linear interpolation based on a defined X-Y node table.



In 14VRS, The "ML_InterpolationLinear (ML_TechBase library)" was renamed to "IL_InterpolationLinear".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-64: Reference table of IL_InterpolationLinear function block

Interface description

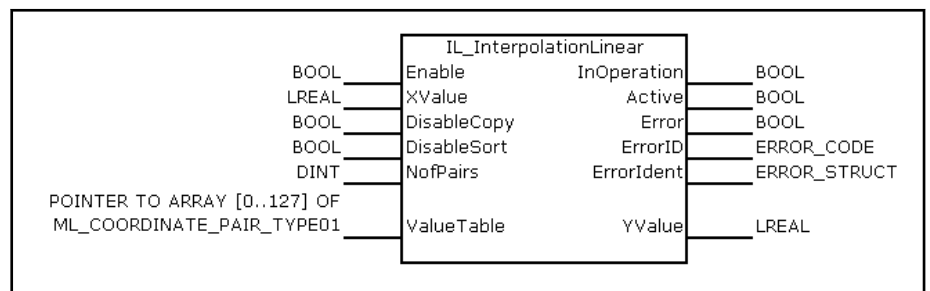


Fig. 4-82: IL_InterpolationLinear function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Enabling the function block (cyclic, state-controlled)
	XValue	LREAL	X value
	DisableCopy	BOOL	Deactivating copying of the data point table
	DisableSort	BOOL	Deactivating sorting of the data point table
	NofPairs	DINT	Number of the connected value pairs in the data point table

I/O type	Name	Data type	Description
	ValueTable	POINTER TO ARRAY [0..127] OF IL_COORDINATE_PAIR_TYPE01	Address of the data point table IL_COORD_PAIR_TYPE01, page 136
VAR_OUTPUT	InOperation	BOOL	Cyclic calculation of a Y value completed without error, output variable "YValue" valid
	Active	BOOL	Calculation of a Y value active, output variables invalid
	Error	BOOL	Calculation of a Y value completed with error, error outputs "ErrorID" and "ErrorIdent" valid
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, the output contains detailed error information Troubleshooting, page 135
	YValue	LREAL	Y value

Tab. 4-65: Interface variables of the IL_InterpolationLinear function block

Functional description

The function block IL_InterpolationLinear cyclically calculates the Y-value "YValue" associated with the input value "XValue" via "Enable" based on a specified data point table after processing enabled for the function block.

The data point table is saved on an internal data array depending on the inputs "DisableCopy" and "DisableSort" at a rising edge and sorted in an ascending order afterwards. The time for calculating an associated y-value "YValue" can be minimized if a data point table already sorted in an ascending order is used when the function block is enabled for processing and this is marked by the input "DisableSort". In this case, the data array is not sorted. If in a new processing of the function block, the internally available and already used data point table should be used again, the input "DisableCopy" as well as the input "DisableSort" have to be set to enabled.

The Y value to be calculated is stored on the output "YValue" in case of error-free processing of the function block, whereupon the validity of the value is signaled by the output "InOperation".

If it is not possible to provide the Y value to be calculated immediately after the activation of the function block, this is indicated by the output "Active".

If an error occurs while processing the function block, this is signaled by the "Error" output. The output "YValue" is not updated in case of an error.

Effect of the inputs "DisableCopy" und "DisableSort" on the runtime behavior of the function block

DisableCopy	DisableSort	Use case
FALSE	FALSE	An unsorted data point table is specified. It was not used like this in the previous calculation. The data point table is copied and sorted
FALSE	TRUE	A data point table already sorted in an ascending order is specified. It was not used like this in the previous calculation. The data point table is copied but not sorted internally

DisableCopy	DisableSort	Use case
TRUE	FALSE	This combination is not a reasonable state
TRUE	TRUE	The data point table was used in previous calculations. Therefore, it is already known internally and sorted in an ascending order

Tab. 4-66: Assignment of the inputs DisableCopy and DisableSort



Sorting the data point table after the activation of the function block might cause that the function block cannot deliver the desired output value in the first or the following cycle due to the restriction to a maximally admissible runtime. This duration at the beginning of the activation is signaled by "Active".

Troubleshooting

The function block uses the error table **MLC_TABLE**, 16#0030. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR (16#0006)	16#F0260030	0	Inputs invalid (XValue not within the data points)
STATE_MACHINE_ERROR (16#0005)	16#F0260031	xx	Invalid state of the state machine. xx indicates the invalid state

Tab. 4-67: Error codes of the IL_InterpolationLinear function block

4.11.3 IL_MaxValue

Brief description

The function block IL_MaxValue determines the actual maximum value from the time dependent course of the input signal.



From 14VRS, the "ML_MaxValue (ML_TechBase library)" function block is now called "IL_MaxValue".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-68: Reference table of the IL_MaxValue function block

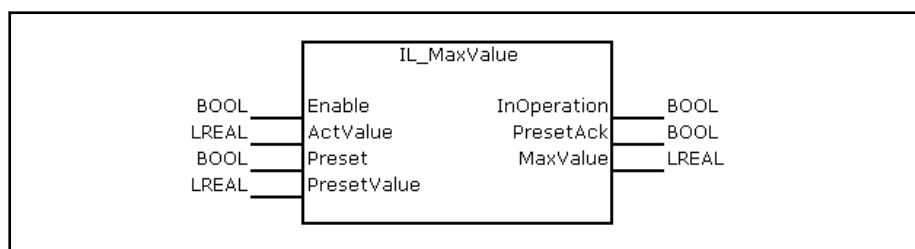
Interface description

Fig. 4-83: IL_MaxValue function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing enabled of the function block (cyclic, state-controlled)
	ActValue	LREAL	Current value
	Preset	BOOL	Activation of preset value
	PresetValue	LREAL	Specified value
VAR_OUTPUT	InOperation	BOOL	Determination of maximum value completed

I/O type	Name	Data type	Description
	PresetAck	BOOL	Confirmation to apply "PresetValue" value
	MaxValue	LREAL	Maximum value

Tab. 4-69: Interface variables of the IL_MaxValue function block

Functional description After the release of the function block, the function block IL_MaxValue uses "Enable" to compare the current value of the input variable "ActValue" to the value in "MaxValue". It copies the respectively higher value to the output variable "MaxValue".

If the input "Preset" is activated at function block runtime, the value of the input variable "PresetValue" is copied to the output variable "MaxValue". The input is state-controlled, i.e. as long as the "Preset" input is active, the "PresetValue" is copied to the "MaxValue" output variable.



If a "PresetValue" value is to be specified when the function is activated, the "Preset" input has to be set simultaneously with the "Enable" input.

Error codes The function block IL_MaxValue does not generate any errors.

4.11.4 IL_COORD_PAIR_TYPE01

This data structure defines a two-dimensional coordinate with X- and Y-LREAL values.

Type declaration

```
TYPE IL_COORD_PAIR_TYPE01 :
STRUCT
  X: LREAL := 0.0;           // X coord
  Y: LREAL := 0.0;           // Y coord
END_STRUCT
END_TYPE
```

Usage in

- [IL_InterpolationLinear](#), page 133

4.12 Character strings without length limitation (StringFun_NoLimitStrLength)

4.12.1 Introduction and overview

Assignment: Target system/library

Target system	Library
MLC 15VRS MTX 15VRS	RIL_Uilities

Tab. 4-70: Reference table

The following functions allow text processing/character string management without length limitation.

Address and length of the character string are applied for example. It is processed byte-by-byte.

The following applies:

Specification 1 Functions to process character strings, such as [IL_LEFT](#), page 155 obtain the two outputs "SourceStr" and "DstStr".

"SourceStr" is the initial address of the source character string.

"DstStr" is the initial address of the target character string (result).

If "SourceStr" and "DstStr" are identical, the source character string is changed and no copy is created. The function returns the address. The result is stored in this address.

Example: IL_LEFT(SourceStr, DstStr, length, MaxLen)

```
Str1 := 'ABCDE';
L_LEFT(ADR(Str1), ADR(Str2), 3, 5);    Return value of the function ADR(Str2)
                                         Str1 = 'ABCDE'
                                         Str2 = 'ABC'
```

```
Str1 := 'ABCDE';
IL_LEFT(ADR (Str1), ADR (Str1), 3, 5);  Return value of the function: ADR(Str1)
                                         Str1 = 'ABC'
```

Specification 2 A character string is accessed via pointer.
The "MaxLen" input limits the character string length to avoid access to invalid memory.
If the maximum number of characters "MaxLen" is reached, "DstStr" is completed with "NUL". The "DstStr" address is returned by the function.

Example: IL_LEFT(SourceStr, DstStr, length, MaxLen)

```
Str1 := 'ABCDEFGHijklmn';
IL_LEFT(ADR(Str1), ADR(Str2), 10, 5);  Return value of the function: ADR(Str2)
                                         Str1 = 'ABCDEFGHijklmn'
                                         Str2 = 'ABCDE'
```

```
Str1 := 'ABCDEFGHijklmn';
IL_LEFT(ADR (Str1), ADR (Str1), 10, 5); Return value of the function: ADR(Str1)
                                         Str1 = 'ABCDE'
```

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Capitalization	
IL_StrLower, page 139	Function converts character string to lower case letters
IL_StrUpper, page 140	Function converts character string to upper case letters
Compare	
IL_StrCompare, page 141	Function compares two character strings and returns the difference. Upper and lower case letters considered
IL_StrCompareNoCase, page 141	Function compares two character strings and returns the difference. Upper and lower case letters not considered

IL_StrNCompare, page 142	Function compares the entered number of characters of two character strings and returns the difference. Upper and lower case letters considered
IL_StrNCompareNoCase, page 143	Function compares the entered number of characters of two character strings and returns the difference. Upper and lower case letters not considered
Conversions	
IL_AsciiStreamToBase64, page 143	Function converts a character string, saved in ASCII format, to the BASE64 format
IL_Base64ToAsciiStream, page 145	Function converts a character string, saved in BASE64 format, to the ASCII format
IL_BufferToString, page 147	Function copies an ASCII character string from a byte array
IL_StringToBuffer, page 148	Function copies an ASCII character string to a byte array
Copy	
IL_StrCopy, page 150	Function copies character string from the source to the target address
IL_StrNCopy, page 151	Function copies partial character string from a character string to a target character string. Length of the target character string is specified. If the source character string is shorter than the requested length, the target character string is supplemented by fill characters
IEC	
IL_CONCAT, page 152	Function adds character string 2 behind character string 1
IL_DELETE, page 153	Function deletes a partial character string from the character string of a specified position
IL_FIND, page 154	Function searches for position of entered partial character string in a character string
IL_INSERT, page 154	Function inserts a partial character string to a character string starting from the entered position
IL_LEFT, page 155	Function provides the left part of a character string with a length of x characters
IL_LEN, page 156	Function provides the character string length
IL_MID, page 156	Function provides a partial character string to a character string starting from the entered position
IL_REPLACE, page 157	Function replaces a partial character string of a character string by another partial character string
IL_RIGHT, page 158	Function provides the right part of a character string with a length of x characters
Numeric conversions (type conversion)	
IL_Ascii_To_Char, page 159	Function converts ASCII code to a character
IL_Char_To_Ascii, page 160	Function converts a character to ASCII code
IL_Int_To_Str, page 160	Function converts (depending on format) an integer to a character string
IL_Real_To_Str, page 161	Function converts (depending on format) a REAL number to a character string
IL_String_To_Unicode, page 162	Function converts a character string according to the specified option (utf8/utf16) into Unicode and stores the values in a DWORD array
IL_Unicode_To_String, page 163	Function converts a Unicode character string to a STRING according to the specified option (utf8/utf16)
Other	

IL_BufferGetElementCount, page 164	The function specifies the number of objects in the character string separated by a "separator" character.
IL_Code, page 165	Function determines the ASCII code of a specified character in a character string in decimal representation
IL_StrInverse, page 166	Function inverts a character string
IL_StrSetEndOfString, page 166	Functions sets the end of a character string
Split	
IL_StrGetElement, page 167	The function is split into multiple elements by the entered separator string. The function calls one of the elements
IL_StrLChr, page 168	Function isolates a partial character string from the character string up to the first occurrence of the entered separator
IL_StrRChr, page 169	Function isolates a partial character string from the character string up to the last occurrence of the entered separator
IL_StrSetElement, page 170	Character string separated into elements by the entered separator order. The function positions a new element to the desired position in the character string
IL_StrSplit, page 171	Character string separated by specified separator. The function stores the address of each partial character string in an array
Trim	
IL_StrTrimAll, page 172	Function deletes all specified characters in a character string
IL_StrTrimSpace, page 173	Depending on the option, the function deletes spaces in a character string

Tab. 4-71: RIL_Uilities, StringFun_NoLimitStrLength

4.12.2 IL_StrLower

Brief description

The **IL_StrLower** function converts all characters of a character string into small letters.

Return value: Address of target character string.

Interface description

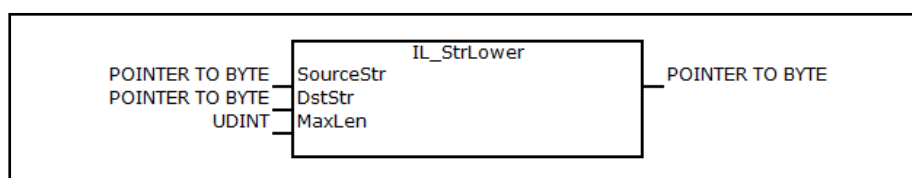


Fig. 4-84: FUN IL_StrLower

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-72: FUN IL_StrLower interface

Example 1

Str1: STRING := 'AbCDE';
Str2: STRING;
IL_StrLower(ADR(Str1), ADR(Str2), 5);

Return value: ADR(Str2)
Str2: 'abcde'

Example 2

Str1: STRING := 'AbCDE';
Str2: STRING;
IL_StrLower(ADR(Str1), ADR(Str2), 2);

Return value: ADR(Str2)
Str2: 'ab'

Example 3

Str1: STRING := 'AbCDE';
Str2: STRING;
IL_StrLower(ADR(Str1), ADR(Str1), 5);

Return value: ADR(Str1)
Str1: 'abcde'

4.12.3 IL_StrUpper

Brief description

The **IL_StrUpper** function converts all characters of a character string into capital letters.
Return value: Address of target character string.

Interface description

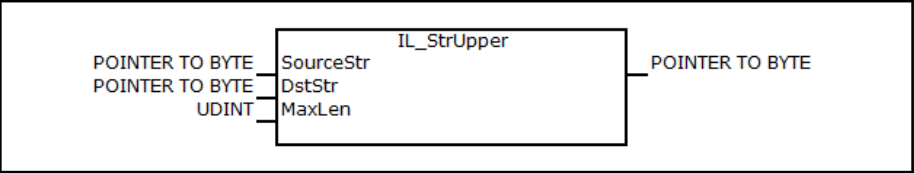


Fig. 4-85: FUN IL_StrUpper

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-73: FUN IL_StrUpper interface

Example 1

Str1: STRING := 'aBcde';
Str2: STRING;
IL_StrUpper(ADR(Str1), ADR(Str2), 5);

Return value: ADR(Str2)
Str2: 'ABCDE'

Example 2

Str1: STRING := 'aBcde';

Str2: STRING;

IL_StrUpper(ADR(Str1), ADR(Str2), 2);

Return value: ADR(Str2)

Str2: 'AB'

Example 3

Str1: STRING := 'aBcde';

IL_StrUpper(ADR(Str1), ADR(Str1), 5);

Return value: ADR(Str1)

Str1: 'ABCDE'

4.12.4 IL_StrCompare**Brief description**

The **IL_StrCompare** function compares two character strings and returns the difference.

Upper and lower case letters considered. The difference is calculated using the first non-identical characters.

If the two character strings are identical, 0 is returned.

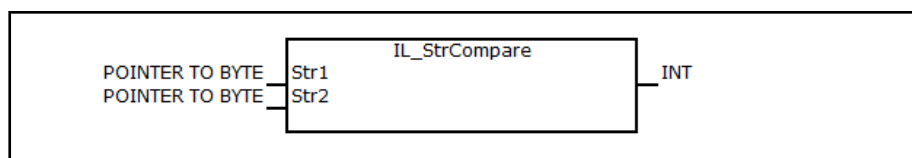
Interface description

Fig. 4-86: FUN IL_StrCompare

Name	Type	Comment
Str1	POINTER TO BYTE	Address of character string 1
Str2	POINTER TO BYTE	Address of character string 2

Tab. 4-74: FUN IL_StrCompare interface

Example

Str1: STRING := 'abcde';

Str2: STRING := 'abCde';

IL_StrCompare (ADR(Str1), ADR(Str2));

Return value: 32

4.12.5 IL_StrCompareNoCase**Brief description**

The **IL_StrCompare** function compares two character strings and returns the difference.

Upper and lower case letters **not** considered. The difference is calculated using the first non-identical characters.

If the two character strings are identical, 0 is returned.

Interface description

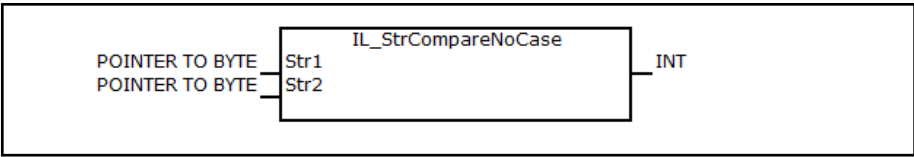


Fig. 4-87: FUN IL_StrCompareNoCase

Name	Type	Comment
Str1	POINTER TO BYTE	Address of character string 1
Str2	POINTER TO BYTE	Address of character string 2

Tab. 4-75: FUN IL_StrCompareNoCase interface

Example

```
Str1: STRING := 'abcde';
Str2: STRING := 'ABEDE'
IL_StrCompareNoCase (ADR(Str1), ADR(Str2));           Return value: -2
```

4.12.6 IL_StrNCompare

Brief description The **IL_StrNCompare** function compares partial character strings of two character strings and returns the difference.
Upper and lower case letters considered.
The difference is calculated using the first non-identical characters.
If the two character strings are identical, 0 is returned.

Interface description

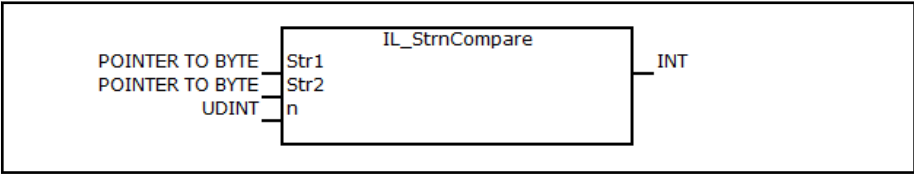


Fig. 4-88: FUN IL_StrNCompare

Name	Type	Comment
Str1	POINTER TO BYTE	Address of character string 1
Str2	POINTER TO BYTE	Address of character string 2
n	UDINT	Number of characters to be compared

Tab. 4-76: FUN IL_StrNCompare interface

Example 1

```
Str1: STRING := 'abcde';
Str2: STRING := 'abCde'
IL_StrNCompare (ADR(Str1), ADR(Str2), 3);           Return value: 32
```

Example 2

Str1: STRING := 'abcde';

Str2: STRING := 'abCde';

IL_StrNCompare (ADR(Str1), ADR(Str2), 2);

Return value: 0

4.12.7 IL_StrNCompareNoCase

Brief description The **IL_StrNCompareNoCase** function compares partial character strings of two character strings and returns the difference.

Upper and lower case letters **not** considered.

The difference is calculated using the first non-identical characters.

If the two character strings are identical, 0 is returned.

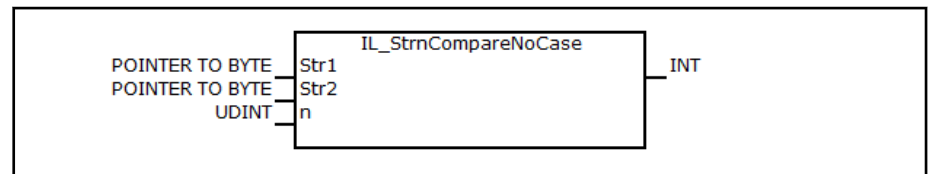
Interface description

Fig. 4-89: FUN IL_StrNCompareNoCase

Name	Type	Comment
Str1	POINTED TO BY BYTE	Address of character string 1
Str2	POINTED TO BY BYTE	Address of character string 2
n	UDINT	Number of characters to be compared

Tab. 4-77: FUN IL_StrNCompareNoCase interface

Example 1

Str1: STRING := 'abcde';

Str2: STRING := 'abEde';

IL_StrNCompareNoCase (ADR(Str1), ADR(Str2), 3);

Return value: -2

Example 2

Str1: STRING := 'abcde';

Str2: STRING := 'abCde';

IL_StrNCompareNoCase (ADR(Str1), ADR(Str2), 3);

Return value: 0

4.12.8 IL_AsciiStreamToBase64

Brief description The **IL_AsciiStreamToBase64** function is used to convert a character string saved in ASCII format to **BASE64** format.

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Utilities

Tab. 4-78: Reference table of the IL_AsciiStreamToBase64

Interface description

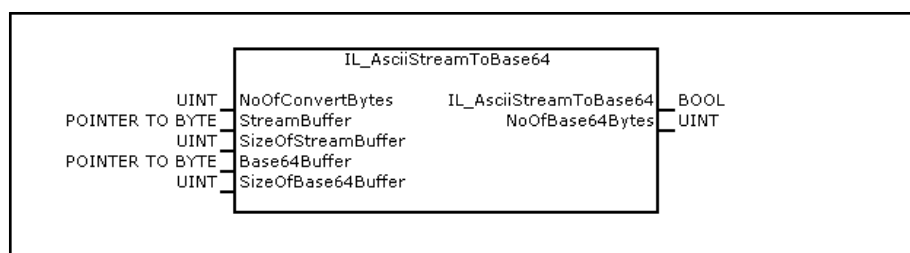


Fig. 4-90: IL_AsciiStreamToBase64 function

	Name	Type	Comment
VAR_INPUT	NoOfConvertBytes	UINT	Number of bytes to be converted in the "StreamBuffer"
	StreamBuffer	POINTER TO BYTE	Pointer to the ASCII character string to be converted Use the ADR () operator
	SizeOfStreamBuffer	UINT	(Maximum) size of the "StreamBuffer" input Use the SIZEOF () operator.
	Base64Buffer	POINTER TO BYTE	Pointer to the buffer of the BASE64-converted data Use the ADR () operator
	SizeOfBase64Buffer	UINT	(Maximum) size of the output BASE64 buffer Use the SIZEOF () operator.
VAR_OUTPUT	NoOfBase64Bytes	UINT	Number of bytes in the BASE64 buffer output as converted result
RETURN		BOOL	Processing is executed.

Tab. 4-79: IL_AsciiStreamToBase64 function

Functional description

The **IL_AsciiStreamToBase64** function is used to convert a character string saved in ASCII format to **BASE64** format.

Source data is transferred to the "StreamBuffer" and the converted data is saved to the "Base64Buffer".

"NoOfConvertBytes" contains the number of bytes to be converted provided in the "StreamBuffer".

The (maximum) size of the "StreamBuffer" is specified in "SizeOfStreamBuffer".

After the conversion, the result is available in "Base64Buffer".

The user has to ensure further processing of "Base64Buffer" data before the next data set is converted.

The (maximum) size of the "Base64Buffer" is specified in "SizeOfBase64Buffer".

The "NoOfBase64Bytes" outputs provides the number of converted and saved bytes in the "Base64Buffer".

The return value "IL_AsciiStreamToBase64" of type BOOL includes information on the success of the function call:

TRUE is output if the conversion was successful.

FALSE is output if data was not completely converted or if "SizeOfStreamBuffer" and "SizeOfBase64Buffer" do not match one another.

Only compatible data is converted.

Examples The **IL_AsciiStreamToBase64** function is used to convert an ASCII character string to BASE64 format.

The return value of type BOOL provides information on the success of the operation.

```
VAR
  bIL_AsciiStreamToBase64: BOOL;
  uiNoOfConvertBytes: UINT := UINT#3;
  arStreamBuffer: STRING(96) := 'Man';
  uiNoOfBase64Bytes : UINT;
  arBase64Buffer : STRING(128);
END_VAR

bIL_AsciiStreamToBase64 := IL_AsciiStreamToBase64(
  NoOfConvertBytes:= uiNoOfConvertBytes,
  StreamBuffer:= ADR(arStreamBuffer),
  SizeOfStreamBuffer:= SIZEOF(arStreamBuffer),
  Base64Buffer:= ADR(arBase64Buffer),
  SizeOfBase64Buffer:= SIZEOF(arBase64Buffer),
  NoOfBase64Bytes=> uiNoOfBase64Bytes);
```

The result of the example "IL_AsciiStreamToBase64" is shown in the following:

bIL_AsciiStreamToBase64	BOOL	TRUE
uiNoOfConvertBytes	UINT	3
arStreamBuffer	STRING(96)	'Man'
uiNoOfBase64Bytes	UINT	4
arBase64Buffer	STRING(128)	'TWFu'

Fig. 4-91: Example "IL_AsciiStreamToBase64"

4.12.9 IL_Base64ToAsciiStream

Brief description

The **IL_Base64ToAsciiStream** function is used to convert a character string saved in **BASE64** format to ASCII format.

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-80: Reference table of the IL_Base64ToAsciiStream

Interface description

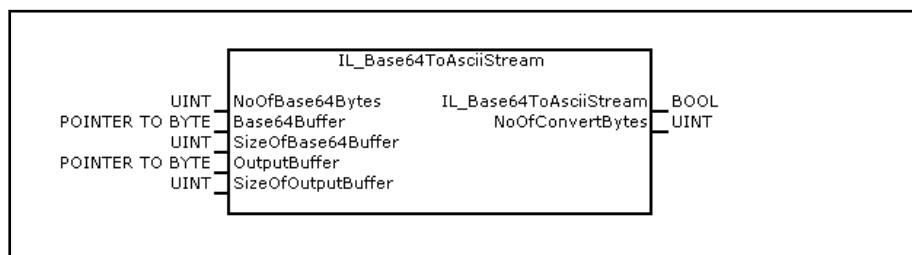


Fig. 4-92: IL_Base64ToAsciiStream function

	Name	Type	Comment
VAR_INPUT	NoOfBase64Bytes	UINT	Number of bytes to be converted in the Base64Buffer
	Base64Buffer	POINTER TO BYTE	Pointer to the BASE64 character string to be converted Use the ADR () operator.

	Name	Type	Comment
	SizeOfBase64Buffer	UINT	(Maximum) size of the "Base64Buffer" input Use the SIZEOF () operator
	OutputBuffer	POINTER TO BYTE	Pointer to the buffer of the ASCII-converted data Use the ADR () operator.
	SizeOfOutputBuffer	UINT	Size of the OutputBuffer Use the SIZEOF () operator
VAR_OUTPUT	NoOfConvertBytes	UINT	Number of bytes in the OutputBuffer output as converted result
RETURN		BOOL	Processing is executed.

Tab. 4-81: IL_Base64ToAsciiStream function

Functional description

The **IL_Base64ToAsciiStream** function is used to convert a character string saved in **BASE64** format to ASCII format.

Source data is transferred to the "Base64Buffer" and the converted data is saved to the "OutputBuffer".

"NoOfBase64Bytes" contains the number of data to be converted provided in the "Base64Buffer".

The (maximum) size of the "Base64Buffer" is specified in "SizeOfBase64Buffer".

After the conversion, the result is available in "OutputBuffer".

The user has to ensure further processing of "OutputBuffer" data before the next data set is converted.

The (maximum) size of the "OutputBuffer" is specified in "SizeOfOutputBuffer".

The "NoOfConvertBytes" outputs provides the number of converted and saved bytes in the "OutputBuffer".

The return value "IL_Base64ToAsciiStream" of type BOOL includes information on the success of the function call:

TRUE is output if the conversion was successful.

FALSE is output if data was not completely converted or if "SizeOfBase64Buffer" and "SizeOfOutputBuffer" do not match one another.

Only compatible data is converted.

Examples

The **IL_Base64ToAsciiStream** function is used to convert a character string saved in **BASE64** format to ASCII format.

The return value of type BOOL provides information on the success of the operation.

```

VAR
  bIL_Base64ToAsciiStream: BOOL;
  uiNoOfBase64Bytes : UINT :=4;
  arBase64Buffer : STRING(128) :='TWFu';
  uiNoOfConvertBytes: UINT;
  arOutputBuffer: STRING(96);
END_VAR

bIL_Base64ToAsciiStream := IL_Base64ToAsciiStream(
  NoOfBase64Bytes:= uiNoOfBase64Bytes,

```

```
Base64Buffer:= ADR(arBase64Buffer),
SizeOfBase64Buffer:= SIZEOF(arBase64Buffer),
OutputBuffer:= ADR(arOutputBuffer),
SizeOfOutputBuffer:= SIZEOF(arOutputBuffer),
NoOfConvertBytes=> uiNoOfConvertBytes);
```

The result of the example "IL_Base64ToAsciiStream" is shown in the following:

bIL_Base64ToAsciiStream	BOOL	TRUE
uiNoOfBase64Bytes	UINT	4
arBase64Buffer	STRING(128)	'TWFu'
uiNoOfConvertBytes	UINT	3
arOutputBuffer	STRING(96)	'Man'

Fig. 4-93: Example "IL_Base64ToAsciiStream"

4.12.10 IL_BufferToString

Brief description

The **IL_BufferToString** function copies an ASCII character string from a byte array.

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-82: Reference table of the IL_BufferToString

Interface description

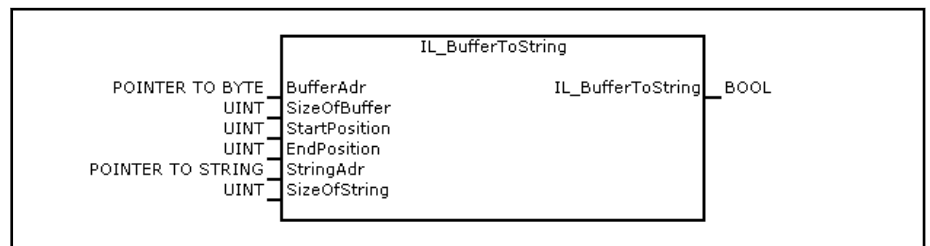


Fig. 4-94: IL_BufferToString

	Name	Type	Comment
VAR_INPUT	BufferAdr	POINTER TO BYTE	(Initial) address of the byte array buffer Use the ADR () operator
	SizeOfBuffer	UINT	Size of the byte array buffer Use the SIZEOF () operator
	StartPosition	UINT	Starting position of the character string in the byte array buffer
	EndPosition	UINT	End position of the character string in the byte array buffer
	StringAdr	POINTER TO STRING	Pointer to copied character string Use the ADR () operator
	SizeOfString	UINT	Maximum permissible size of the character string Use the SIZEOF () operator SizeOf(StringAdr)
RETURN		BOOL	Conversion completed

Tab. 4-83: IL_BufferToString function

Functional description

The **IL_BufferToString** function copies an ASCII character string from a byte array.

"BufferAdr" contains the (starting) address of the provided byte array.

"StartPosition" includes the starting position of the character string in the array.

"EndPosition" contains the position of the latest character to be read in the array.

The function queries a character string from a byte array starting at "StartPosition" and ending at "EndPosition".

The retrieved character string is saved under "StringAdr".

The maximum length of the retrievable character string is specified under "SizeOfString".

The return value of the "IL_BufferToString" function of type BOOL confirms the conversion.

TRUE is output if copying was successful.

FALSE is output if copying was not successful, e.g. if "SizeOfBuffer" and "SizeOfString" do not match.

Only compatible data is copied.

Examples

The IL_BufferToString function copies an ASCII character string from a byte array.

```
VAR
  bIL_BufferToString : BOOL;
  strIL_BufferToString: STRING(300);
  arBufferAdr: ARRAY [0..300] OF BYTE:= [63,64,65,66,67,68,69,70,71,72,73];
END_VAR

bIL_BufferToString := IL_BufferToString(BufferAdr:= ADR(arBufferAdr),
  SizeOfBuffer:=SIZEOF(arBufferAdr), StartPosition:=2, EndPosition:=8,
  StringAdr:= ADR(strIL_BufferToString),
  SizeOfString:= SIZEOF(strIL_BufferToString));
```

The result of the example "IL_BufferToString" is shown in the following:

bIL_BufferToString	BOOL	TRUE
strIL_BufferToString	STRING(300)	'ABCDEFG'
arBufferAdr	ARRAY [0..300] OF BYTE	
arBufferAdr[0]	BYTE	63
arBufferAdr[1]	BYTE	64
arBufferAdr[2]	BYTE	65
arBufferAdr[3]	BYTE	66
arBufferAdr[4]	BYTE	67
arBufferAdr[5]	BYTE	68
arBufferAdr[6]	BYTE	69
arBufferAdr[7]	BYTE	70
arBufferAdr[8]	BYTE	71
arBufferAdr[9]	BYTE	72
arBufferAdr[10]	BYTE	73
arBufferAdr[11]	BYTE	0

Fig. 4-95: Example "IL_BufferToString"

4.12.11 IL_StringToBuffer

Brief description

The IL_StringToBuffer function copies an ASCII character string to a byte array.

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-84: Reference table of the IL_StringToBuffer

Interface description

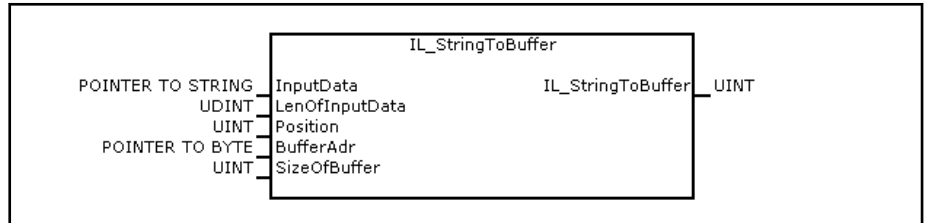


Fig. 4-96: IL_StringToBuffer function

	Name	Type	Comment
VAR_INPUT	InputData	POINTER TO STRING	Pointer to the character string to be converted Use the ADR () operator
	LenOfInputData	UDINT	Length of the input character string Use IL_Len (InputData)
	Position	UINT	Starting position of the character string in the byte array buffer
	BufferAdr	POINTER TO BYTE	Address of the byte array buffer Use the ADR () operator
	SizeOfBuffer	UINT	Size of the byte array buffer. Use the SIZEOF () operator
RETURN		UINT	End position of the character string in the byte array buffer

Tab. 4-85: IL_StringToBuffer function

Functional description

The **IL_StringToBuffer** function copies an ASCII character string to a byte array.

"InputData" contains the pointer to the character string to be copied.

The number of characters is specified in "LenOfInputData". It is to be determined using the "IL_Len" function.

The function also retrieves the character string and stores it in the "BufferAdr" array.

"Position" is the starting position for the storage of the character string in the "BufferAdr" array.

The first character in the "BufferAdr" array is provided with the position number 0.

The return value "IL_StringToBuffer" of type UINT returns the position of the latest copied character.

In case of error, if e.g. "SizeOfBuffer" and "LenOfInputData" do not match, copying is canceled.

Only compatible data is copied.

Examples

The **IL_StringToBuffer** function copies an ASCII character string to a byte array.

```
VAR
  strInputData: STRING(300) :='ABCDEFGH';
  uiIL_StringToBuffer: UINT;
  arBufferAdr: ARRAY [0..300] OF BYTE;
END_VAR

uiIL_StringToBuffer := IL_StringToBuffer(InputData:=ADR(strInputData),
  LenOfInputData := IL_LEN(ADR(strInputData)), Position:=4,
  BufferAdr:=ADR(arBufferAdr) , SizeOfBuffer:= SIZEOF(arBufferAdr));
```

The result of the example "IL_StringToBuffer" is shown in the following:

strInputData	STRING(300)	'ABCDEFGH'
uiIL_StringToBuffer	UINT	11
arBufferAdr	ARRAY [0..300]...	
arBufferAdr[0]	BYTE	0
arBufferAdr[1]	BYTE	0
arBufferAdr[2]	BYTE	0
arBufferAdr[3]	BYTE	0
arBufferAdr[4]	BYTE	65
arBufferAdr[5]	BYTE	66
arBufferAdr[6]	BYTE	67
arBufferAdr[7]	BYTE	68
arBufferAdr[8]	BYTE	69
arBufferAdr[9]	BYTE	70
arBufferAdr[10]	BYTE	71
arBufferAdr[11]	BYTE	72
arBufferAdr[12]	BYTE	0
arBufferAdr[13]	BYTE	0
arBufferAdr[14]	BYTE	0

Fig. 4-97: Example "IL_StringToBuffer"

4.12.12 IL_StrCopy

Brief description The **IL_StrCopy** function copies a character string from source to target address.

Return value: Address of target character string.

Interface description

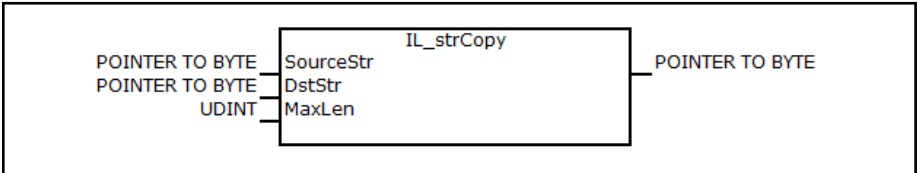


Fig. 4-98: FUN IL_StrCopy

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-86: FUN IL_StrCopy interface

Example

Str1: STRING := 'abcde';

Str2: STRING;

IL_StrCopy (ADR(Str1), ADR(Str2), 5);

Return value: ADR(Str2)

Str2: 'abcde'

4.12.13 IL_StrNCopy**Brief description**

The **IL_StrNCopy** function copies a partial character string from a character string to the target address.

Length of the target character string is specified with "MaxLen".

If the source character string is shorter than the entered length, the target character string is supplemented by fill characters

Return value: Address of target character string.

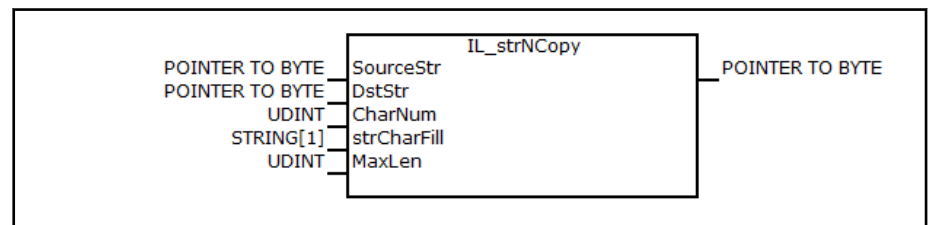
Interface description

Fig. 4-99: FUN IL_StrNCopy

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
CharNum	UDINT	Length of the target character string (in bytes/Char)
strCharFill	STRING[1]	Fill character
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-87: FUN IL_StrNCopy interface

Example 1

Str1: STRING := 'abcde';

Str2: STRING;

IL_StrNCopy (ADR(Str1), ADR(Str2), 7, 'F' , 10);

Return value: ADR(Str2)

Str2: 'abcdeFF'

Example 2

Str1: STRING := abcde;

Str2: STRING;

IL_StrNCopy (ADR(Str1), ADR(Str2), 7, 'F', 6);

Return value: ADR(Str2)

Str2: 'abcdeF'

Example 3

```
Str1: STRING := abcde;
Str2: STRING;
IL_StrNCopy (ADR(Str1), ADR(Str2), 3, 'F',10);
```

Return value: ADR(Str2)
Str2: 'abc'

4.12.14 IL_CONCAT

Brief description The **IL_CONCAT** function adds character string 2 behind character string 1.

 The result **cannot** be saved under the address of character string 2.

Return value: Address of target character string.

Interface description

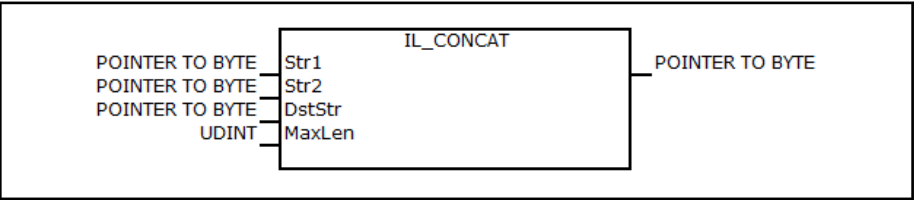


Fig. 4-100: FUN IL_CONCAT

Name	Type	Comment
Str1	POINTER TO BYTE	Address of character string 1
Str2	POINTER TO BYTE	Address of character string 2
DstStr	POINTER TO BYTE	Address of target character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-88: FUN IL_CONCAT interface

Example 1

```
Str1: STRING := 'ABCD';
Str2: STRING := 'EFGH';
Str3: STRING;
IL_CONCAT(ADR(Str1), ADR(Str2), ADR(Str3), 10);
```

Return value: ADR(Str3)
Str3: 'ABCDEFGH'

Example 2

```
Str1: STRING := 'ABCD';
Str2: STRING := 'EFGH';
Str3: STRING;
IL_CONCAT(ADR(Str1), ADR(Str2), ADR(Str3), 2);
```

Return value: ADR(Str3)
Str3: 'AB'

Example 3

Str1: STRING := 'ABCD';

Str2: STRING := 'EFGH';

IL_CONCAT(ADR(Str1), ADR(Str2), ADR(Str1), 10);

Return value: ADR(Str1)

Str1: 'ABCDEFGH'

4.12.15 IL_DELETE**Brief description**

The **IL_DELETE** function deletes a partial character string from a character string from a specified position.

The first character of the character string counted from the left is on position 1.

Return value: Address of target character string.

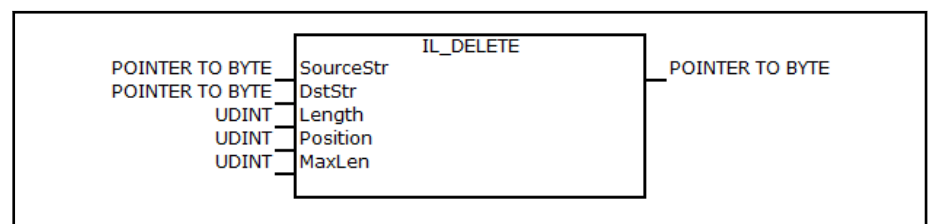
Interface description

Fig. 4-101: FUN IL_DELETE

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Length	UDINT	Length of the partial character string to be removed (in bytes/Char)
Position	UDINT	Position of the source character string to start the removal of the partial character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-89: FUN IL_DELETE interface

Example 1

Str1: STRING := 'ABCDEFGH';

Str2: STRING;

IL_DELETE (ADR(Str1), ADR(Str2), 4, 3, 5);

Return value: ADR(Str2)

Str2: 'ABGH'

Example 2

Str1: STRING := 'ABCDEFGH';

Str2: STRING;

IL_DELETE (ADR(Str1), ADR(Str2), 4, 3, 2);

Return value: ADR(Str2)

Str2: 'AB'

Example 3

```
Str1: STRING := 'ABCDEFGH';  
IL_DELETE (ADR(Str1), ADR(Str1), 4, 3, 5);  
Return value: ADR(Str1)  
Str1: 'ABGH'
```

4.12.16 IL_FIND

Brief description The **IL_FIND** searches for the initial position of the specified partial character string in a character string. The first character of the character string counted from the left is on position 1.

The position is provided as return of the function.

Interface description

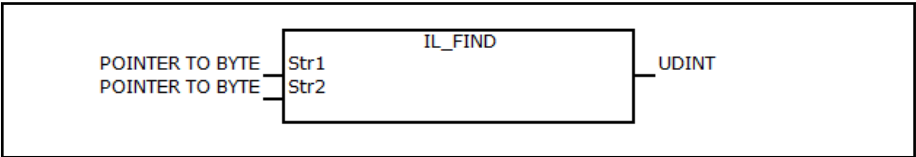


Fig. 4-102: FUN IL_FIND

Name	Type	Comment
Str1	POINTER TO BYTE	Address of source character string
Str2	POINTER TO BYTE	Address of the partial character string to be searched for

Tab. 4-90: FUN IL_FIND interface

Example

```
Str1: STRING := 'ABCDEFGH';  
Str2: STRING := 'CDEF';  
IL_FIND (ADR(Str1), ADR(Str2));  
Return value: 3
```

4.12.17 IL_INSERT

Brief description The **IL_INSERT** inserts a partial character string starting from a specified position in the character string.

The first character of the character string counted from the left is on position 1.

Return value: Address of the target character string.

Interface description

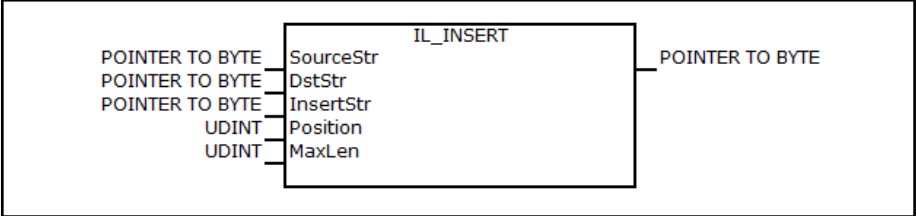


Fig. 4-103: FUN IL_INSERT

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
InsertStr	POINTER TO BYTE	Address of the partial character string to be inserted

Name	Type	Comment
Position	UDINT	Position of the source character string to start the insertion of the partial character string from
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-91: FUN IL_INSERT interface

Example 1

```

Str1: STRING := 'ABCD';
Str2: STRING;
Str3: STRING := 'EFGH';
IL_INSERT(ADR(Str1), ADR(Str2), ADR(Str3), 3, 10);    Return value: ADR(Str2)
                                                    Str2: 'ABEFGHCD'

```

Example 2

```

Str1: STRING := 'ABCD';
Str2: STRING;
Str3: STRING := 'EFGH';
IL_INSERT(ADR(Str1), ADR(Str2), ADR(Str3), 3, 2);    Return value: ADR(Str2)
                                                    Str2: 'AB'

```

Example 3

```

Str1: STRING := 'ABCD';
Str3: STRING := 'EFGH';
IL_INSERT(ADR(Str1), ADR(Str1), ADR(Str3), 3, 10);    Return value: ADR(Str1)
                                                    Str1: 'ABEFGHCD'

```

4.12.18 IL_LEFT

Brief description The **IL_LEFT** provides the left partial character string of a character string.
Return value: Address of target character string.

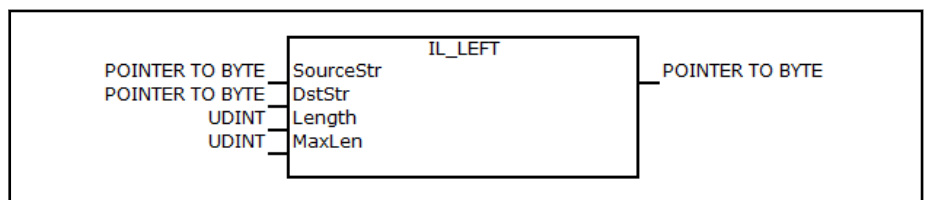
Interface description

Fig. 4-104: IL_LEFT

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string

Name	Type	Comment
Length	UDINT	Length of the left partial character string (in Byte/Char)
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-92: FUN IL_LEFT interface

Example 1

Str1: STRING := 'ABCD';
Str2: STRING;
IL_LEFT(ADR(Str1), ADR(Str2), 3, 5);

Return value: ADR(Str2)
Str2: 'ABC'

Example 2

Str1: STRING := 'ABCD';
Str2: STRING;
IL_LEFT(ADR(Str1), ADR(Str2), 3, 2);

Return value: ADR(Str2)
Str2: 'AB'

Example 3

Str1: STRING := 'ABCD';
IL_LEFT(ADR (Str1), ADR (Str1), 3, 5);

Return value: ADR(Str1)
Str1: 'ABC'

4.12.19 IL_LEN

Brief description The **IL_LEN** function provides the length of the character string (number of characters).

Interface description

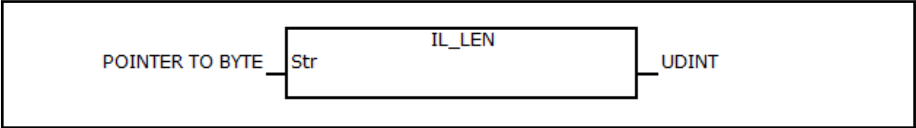


Fig. 4-105: FUN IL_LEN

Name	Type	Comment
Str	POINTER TO BYTE	Address of character string

Tab. 4-93: FUN IL_LEN interface

Example

Str1: STRING := 'ABCDEFGH';
IL_LEN(ADR(Str1));

Return value: 8

4.12.20 IL_MID

Brief description The **IL_MID** functions provides a partial character string of specified length starting from the specified position of a character string.

The position corresponds to the character number. The first character of the character string counted from the left is on position 1.

Return value: Address of target character string.

Interface description

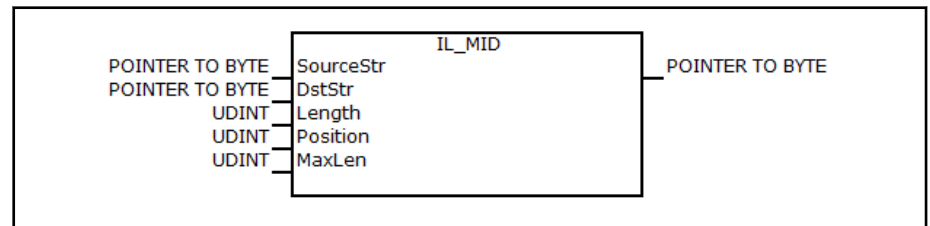


Fig. 4-106: FUN IL_MID

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Length	UDINT	Length of the partial character string (in bytes/Char)
Position	UDINT	Position of the source character string to start calling the partial character string from
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-94: FUN IL_MID interface

Example 1

Str1: STRING := 'ABCDEFGH';

Str2: STRING;

IL_MID(ADR(Str1), ADR(Str2), 4, 3, 10);

Return value: ADR(Str2)

Str2: 'CDEF'

Example 2

Str1: STRING := 'ABCDEFGH';

Str2: STRING;

IL_MID(ADR(Str1), ADR(Str2), 4, 3, 2);

Return value: ADR(Str2)

Str2: 'CD'

Example 3

Str1: STRING := 'ABCDEFGH';

IL_MID(ADR(Str1), ADR(Str1), 4, 3, 10);

Return value: ADR(Str1)

Str1: 'CDEF'

4.12.21 IL_REPLACE

Brief description

The **IL_REPLACE** replaces a partial character string of a character string with a substitute character string.

Return value: Address of target character string.

Interface description

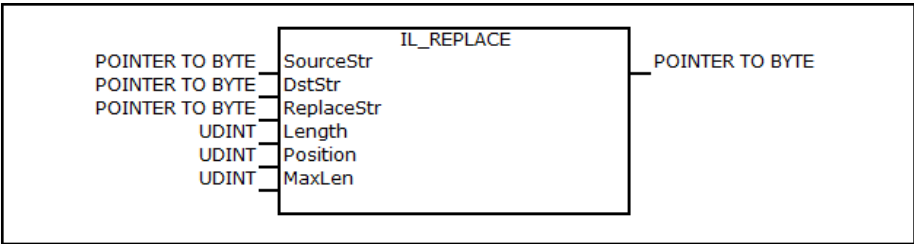


Fig. 4-107: FUN IL_REPLACE

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
ReplaceStr	POINTER TO BYTE	Address of the partial character string to be inserted (substitute character string)
Length	UDINT	Length of the partial character string to be removed (in bytes/Char)
Position	UDINT	The position of the source character string from which the partial character string is to be replaced by the substitute character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-95: FUN IL_REPLACE interface

Example 1

```
Str1: STRING := 'ABCDE';
Str2: STRING;
Str3: STRING := 'XX';
IL_REPLACE(ADR(Str1), ADR(Str2), ADR(Str3), 3, 2, 5);    Return value: ADR(Str2)
                                                         Str3: 'AXXE'
```

Example 2

```
Str1: STRING := 'ABCDE';
Str2: STRING;
Str3: STRING := 'XX';
IL_REPLACE(ADR(Str1), ADR(Str2), ADR(Str3), 3, 2, 2);    Return value: ADR(Str2)
                                                         Str3: 'AX'
```

Example 3

```
Str1: STRING := 'ABCDE';
Str3: STRING := 'XX';
IL_REPLACE(ADR(Str1), ADR(Str1), ADR(Str3), 3, 2, 5);    Return value: ADR(Str1)
                                                         Str1: 'AXXE'
```

4.12.22 IL_RIGHT

Brief description The **IL_RIGHT** provides the right partial character string of a character string.
Return value: Address of target character string.

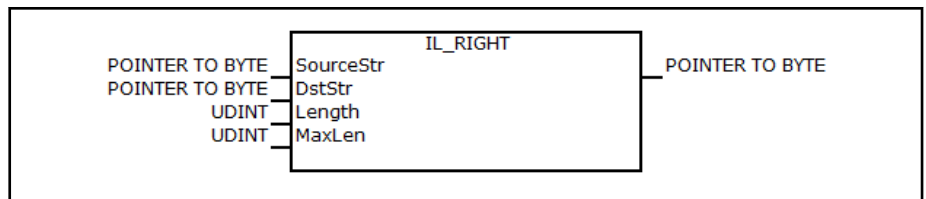
Interface description

Fig. 4-108: FUN IL_RIGHT

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Length	UDINT	Length of the right partial character string (in bytes/Char)
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-96: FUN IL_RIGHT interface

Example 1

Str1: STRING := 'ABCD';

Str2: STRING;

IL_RIGHT(ADR(Str1), ADR(Str2), 3, 5);

Return value: ADR(Str2)

Str2: 'BCD'

Example 2

Str1: STRING := 'ABCD';

Str2: STRING;

IL_RIGHT(ADR(Str1), ADR(Str2), 3, 2);

Return value: ADR(Str2)

Str2: 'BC'

Example 3

Str1: STRING := 'ABCD';

IL_RIGHT(ADR(Str1), ADR(Str1), 3, 5);

Return value: ADR(Str1)

Str1: 'BCD'

4.12.23 IL_Ascii_To_Char**Brief description**The **IL_Ascii_To_Char** function converts ASCII code to a character.

Return value: Character

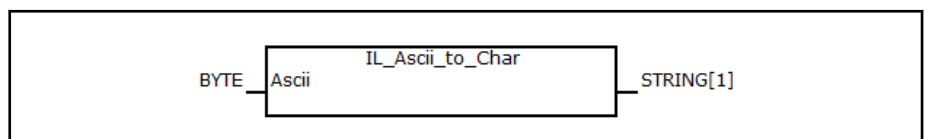
Interface description

Fig. 4-109: FUN IL_Ascii_To_Char

Name	Type	Comment
Ascii	BYTE	ASCII code to be converted

Tab. 4-97: FUN IL_Ascii_To_Char interface

Example

IL_Ascii_to_Char (16#63);	Return value: 'c'
IL_Ascii_to_Char (99);	Return value: 'c'

4.12.24 IL_Char_To_Ascii

Brief description The **IL_Char_To_Ascii** function converts a character to ASCII.
Return value: ASCII code.

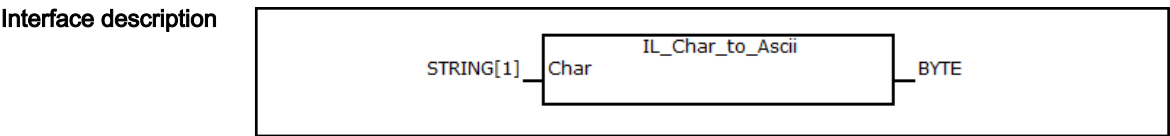


Fig. 4-110: FUN IL_Char_To_Ascii

Name	Type	Comment
Char	STRING[1]	Character to be converted

Tab. 4-98: FUN IL_Char_To_Ascii interface

Example

IL_Char_To_Ascii ('c')	Return value: 16#63
IL_Char_To_Ascii ('c')	Return value: 99

4.12.25 IL_Int_To_Str

Brief description The **IL_Int_To_Str** functions converts an integer (depending on format) to a character string.
Return value: Character string.

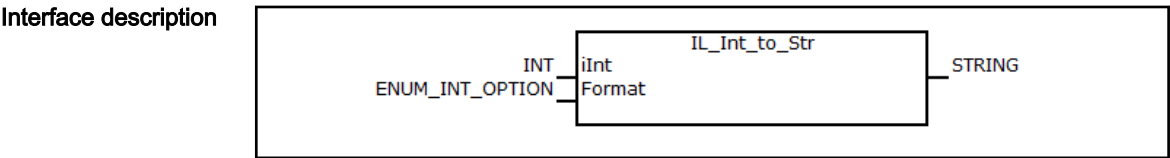


Fig. 4-111: FUN IL_Int_To_Str

Name	Type	Comment
iInt	INT	Integer to be converted
Format	ENUM_INT_OPTION	Selection of digit representation in the character string ENUM_INT_OPTION, page 161

Tab. 4-99: FUN IL_Int_To_Str interface

Example 1

IL_Int_To_Str (10, EN_OPT_BIN);	Return value: '1010'

Example 2

IL_Int_To_Str (10, EN_OPT_DEZ);	Return value: '10'

Example 3

IL_Int_To_Str (10, EN_OPT_HEX);

Return value: 'A'

4.12.26 ENUM_INT_OPTION

The enumeration type **ENUM_INT_OPTION** allows to select the representation of the integer in the character string.

Name	Value	Description
EN_OPT_BIN	0	Binary number
EN_OPT_DEZ	1	Decimal number
EN_OPT_HEX	2	Hexadecimal number

Tab. 4-100: ENUM_INT_OPTION elements

4.12.27 IL_Real_To_Str**Brief description**

The **IL_Real_To_Str** function converts (depending on format) a REAL number (max. seven significant positions) to a character string.

Return value: Address of target character string.

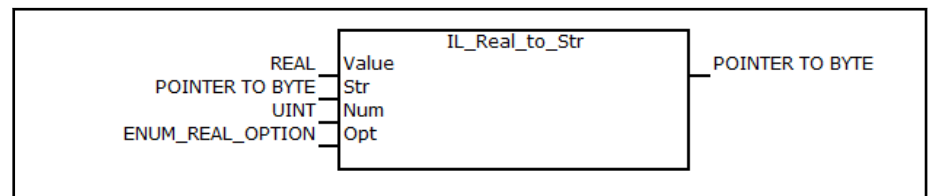
Interface description

Fig. 4-112: FUN IL_Real_To_Str

Name	Type	Comment
Value	REAL	REAL number to be converted
Str	POINTER TO BYTE	Address of target character string
Num	UINT	Number of decimal places to be displayed
Opt	ENUM_REAL_OPTION	Selection of digit representation in the character string ENUM_REAL_OPTION, page 162

Tab. 4-101: FUN IL_Real_To_Str interface

Example 1

IL_Real_To_Str (263.355, 2, EN_OPT_FLOATING);

Return value: '263.36'

Example 2

IL_Real_To_Str (263.355, 2, EN_OPT_EXP);

Return value: '2.63E+2'

Example 3

IL_Real_To_Str (263.355, 0, EN_OPT_ENGINEERING);

Return value: '263*10^0'

IL_Real_To_Str (263.355, 2, EN_OPT_ENGINEERING);

Return value: '2.63*10^+3'

4.12.28 ENUM_REAL_OPTION

The enumeration type **ENUM_REAL_OPTION** allows to select the representation of the REAL number in the character string.

Name	Value	Description
EN_OPT_FLOATING	0	Floating point representation
EN_OPT_EXP	1	Exponential representation
EN_OPT_ENGINEERING	2	Engineer representation

Tab. 4-102: ENUM_REAL_OPTION elements

4.12.29 IL_String_To_Unicode

Brief description The **IL_String_To_Unicode** function converts a character string according to the specified option (utf8/utf16) into Unicode and stores the values in a DWORD array.

 Only 1 byte of memory space is reserved for each string character. Thus, the characters matching ASCII code and Windows 1252 can be identified.

Interface description

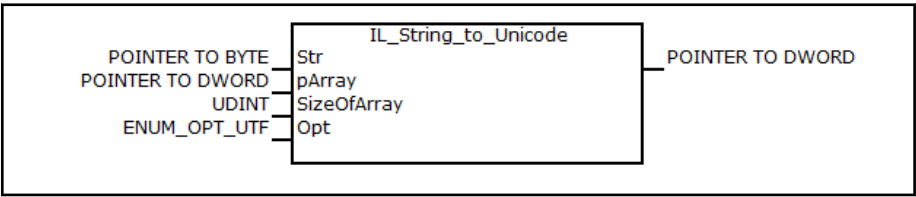


Fig. 4-113: FUN IL_String_To_Unicode

Name	Type	Comment
Str	POINTERS TO BYTE	Address of source character string
pArray	POINTERS TO DWORD	Address of target array
SizeOfArray	UDINT	Max. size of array (in bytes)
Opt	ENUM_OPT_UTF	Unicode coding format ENUM_OPT_UTF, page 163

Tab. 4-103: FUN IL_String_To_Unicode interface

Example 1

Str1: STRING := 'ABYæ';
arArray: Array[0..3] OF DWORD;
IL_String_To_Unicode (ADR(Str1), ADR(arArray), 4, EN_UTF8);

Return value: ADR(arArray)
arArray[0]: 16#41
arArray[1]: 16#42
arArray[2]: 16#E6
arArray[3]: 16#DD

Example 2

Str1: STRING := 'ABŸæ';

arArray: Array[0..3] OF DWORD;

IL_String_To_Unicode (ADR(Str1), ADR(arArray), 4, EN_UTF16);

Return value: ADR(arArray)

arArray[0]: 16#41

arArray[1]: 16#42

arArray[2]: 16#C39D

arArray[3]: 16#C3A6

4.12.30 ENUM_OPT_UTF

The enumeration type **ENUM_OPT_UTF**, type INT, allows to select the Unicode coding format.

Name	Value	Description
EN_UTF16	0	UTF16 code
EN_UTF8	1	UTF8 code

Tab. 4-104: *ENUM_OPT_UTF elements*

4.12.31 IL_Unicode_To_String

Brief description The **IL_Unicode_To_String** function converts Unicode to STRING.

Unicode is stored in a DWORD array. One array element corresponds to one Unicode character.



Only 1 byte of memory space is reserved for each string character. Thus, the characters matching ASCII code and Windows 1252 can be identified.

Return value: Address of target character string.

Interface description

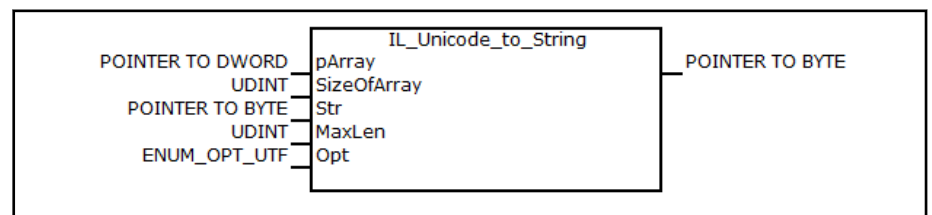


Fig. 4-114: *FUN IL_Unicode_To_String*

Name	Type	Comment
pArray	POINTER TO DWORD	Address of the source array
SizeOfArray	UDINT	Max. size of array (in bytes)
Str	POINTER TO BYTE	Address of target character string
Opt	ENUM_OPT_UTF	Unicode coding format ENUM_OPT_UTF, page 163

Tab. 4-105: *FUN IL_Unicode_To_String interface*

Example 1

arArray: Array[0..3] of DWORD := [16#41, 16#42, 16#E6, 16#DD];

Str1: STRING;

IL_Unicode_To_String (ADR(arArray), 4, ADR(Str1), 4, EN_UTF8);

Return value: ADR(Str1)

Str1: 'ABÝæ'

Example 2

arArray: Array[0..3] of DWORD := [16#41, 16#42, 16#C39D, 16#C3A6];

Str1: STRING;

IL_String_To_Unicode (ADR(Str1), ADR(arArray), 4, EN_UTF16);

Return value: ADR(Str1)

Str1: 'ABÝæ'

4.12.32 IL_BufferGetElementCount**Brief description**

The **IL_BufferGetElementCount** function specifies the number of objects in the character string separated by a "separator" character.

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-106: Reference table of the **IL_BufferGetElementCount**

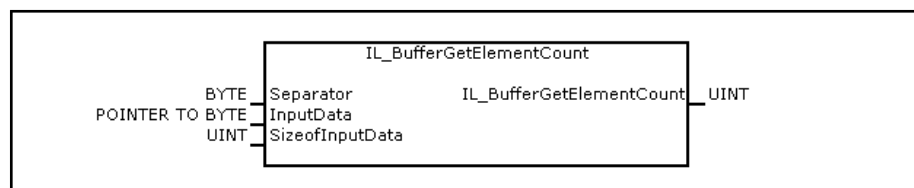
Interface description

Fig. 4-115: **IL_BufferGetElementCount** function

	Name	Type	Comment
VAR_INPUT	Separator	BYTE	ASCII separator in the character string to be analyzed
	InputData	POINTER TO BYTE	Pointer to the character string to be converted Use the ADR() operator
	SizeOfInputData	UINT	Total length of the character string Use the function IL_Len() and the operator ADR() for the character string to be analyzed
RETURN		UINT	Number of separated objects

Tab. 4-107: **IL_BufferGetElementCount** function

Functional description

The **IL_BufferGetElementCount** function specifies the number of objects in the character string separated by a "separator" character.

Example: Character string: '0/123/2/3', separator: /, return value: 4.

If there is only one character in the "InputData" character string, no "separator" is required.

1 is output as return value of the function, e.g. ('X',/) = 1.

If "InputData" is an empty character string, 0 is output as return value of the function, e.g. ("",/) = 0.

Examples The **IL_BufferGetElementCount** function specifies the number of objects in the character string separated by a "separator" character.

```
VAR
  strInputString:   STRING(300) ;
  uiNoOfCharacters: UINT;
END_VAR

strInputString := '1/2/333/4/a/b/C/D/!/@/</>;/,{/[';

uiNoOfCharacters := IL_BufferGetElementCount(
  Separator:= IL_AsciiToByte('/'),
  InputData:= ADR(strInputString),
  SizeOfInputData := UDINT_TO_UINT (IL_Len(ADR(strInputString))));
```

The result of the example "IL_BufferGetElementCount" is shown in the following:

strInputString	STRING(300)	'1/2/333/4/a/b/C/D/!/@/</>;/,{/[	
uiNoOfCharacters	UINT	15	

Fig. 4-116: Example "IL_BufferGetElementCount"

4.12.33 IL_Code

Brief description

The **IL_Code** function determines the ASCII code of a specified character in a character string in decimal representation

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities

Tab. 4-108: Reference table of the IL_Code

Interface description

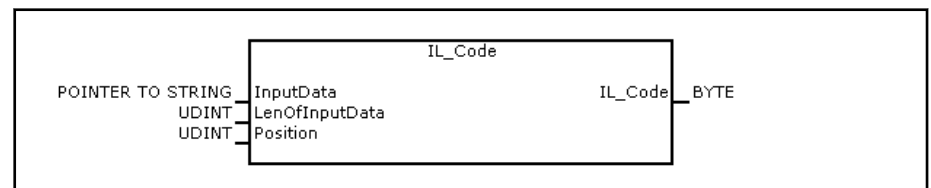


Fig. 4-117: IL_Code function

	Name	Type	Comment
VAR_INPUT	InputData	POINTER TO STRING	Pointer to an ASCII character string Use the ADR() operator
	LenOfInputData	UDINT	String length Use: IL_Len (InputData)
	Position	UDINT	Position at which the character is read
RETURN		BYTE	Code of the character at the "Position"

Tab. 4-109: IL_Code function

Functional description

The **IL_Code** function determines the ASCII code of a specified character in a character string in decimal representation

The position of the character is specified by the "Position" value. The first character of the character string has the position 1.

If **IL_Code** is called with a "position" smaller than 1 or greater than the length of the "LenOfInputData" character string, 0 returns.

Calculate "LenOfInputData" using the IL_Len function.

Example:

IL_Code ('ABC 123', 4) results in 32 (the blank is coded with the value 32).
The return value of type BYTE returns the character as ASCII value.

Examples The **IL_Code** function determines the ASCII code in decimal representation for the character at the position "Position" of a character string.

```
VAR
  strCode: STRING(1000) := 'ABCDEF';
  byIL_Code: BYTE;
END_VAR

byIL_Code := IL_Code(InputData:= ADR(strCode),
  LenOfInputData := IL_Len(ADR(strCode)), Position:=3 );
```

The result of the example "IL_Code" is shown in the following:

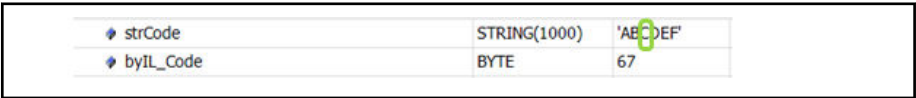


Fig. 4-118: Example "IL_Code"

4.12.34 IL_StrInverse

Brief description The **IL_StrInverse** function inverts a character string.
Return value: Address of target character string.

Interface description

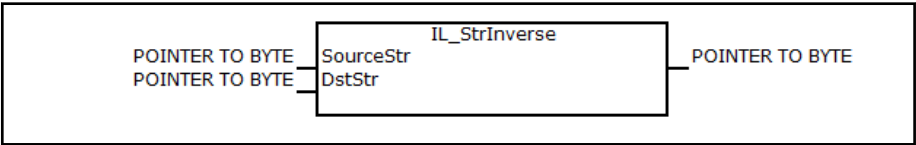


Fig. 4-119: FUN IL_StrInverse

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string

Tab. 4-110: FUN IL_StrInverse interface

Example 1

```
Str1: STRING := 'abcde';
Str2: STRING;

IL_StrInverse (ADR(Str1), ADR(Str2));
```

Return value: ADR(Str2)
Str2: 'edcba'

Example 2

```
Str1: STRING := 'abcde';

IL_StrInverse (ADR(Str1), ADR(Str1));
```

Return value: ADR(Str1)
Str1: 'edcba'

4.12.35 IL_StrSetEndOfString

Brief description The **IL_StrSetEndOfString** functions sets the end ID of a character string and thus limits the length of the target character string.
Return value: Address of target character string.

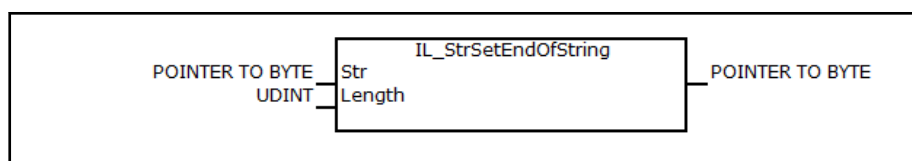
Interface description

Fig. 4-120: FUN IL_StrSetEndOfString

Name	Type	Comment
Str	POINTER TO BYTE	Address of source character string
Length	UDINT	Length of the target character string

Tab. 4-111: FUN IL_StrSetEndOfString interface

Example

Str1: STRING := 'abcde';

IL_StrSetEndOfString (ADR(Str1), 3);

Return value: ADR(Str1)

Str1: 'abc'

4.12.36 IL_StrGetElement

Brief description The source character string is split into elements by separator character strings.

The **IL_StrGetElement** function calls the element specified by "ElementNum".

Return value: Address of the element.

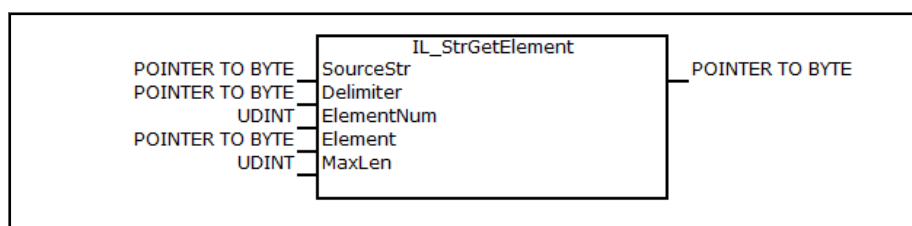
Interface description

Fig. 4-121: FUN IL_StrGetElement

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
Delimiter	POINTER TO BYTE	Address of the target character string
ElementNum	UDINT	Number of the element to be called. The number of the first element is 1
Element	POINTER TO BYTE	Address used to store the called element
MaxLen	UDINT	Max. length of the element (in bytes/Char)

Tab. 4-112: FUN IL_StrGetElement interface

Example 1

Str1: STRING := 'ab;;cd;;e';
Str2: STRING := ';;' ;
Str3: STRING;
IL_StrGetElement (ADR(Str1), ADR(Str2), 2 , ADR(Str3), 5);

Return value: ADR(Str3)
Str3: 'cd'

Example 2

Str1: STRING := 'ab;;cd;;e';
Str2: STRING := ';;' ;
Str3: STRING;
IL_StrGetElement (ADR(Str1), ADR(Str2), 2 , ADR(Str3), 1);

Return value: ADR(Str3)
Str3: 'cd'

Example 3

Str1: STRING := 'ab;;cd;;e';
Str2: STRING := ';;' ;
IL_StrGetElement (ADR(Str1), ADR(Str2), 2 , ADR(Str1), 5);

Return value: ADR(Str1)
Str1: 'cd'

4.12.37 IL_StrLChr

Brief description

The **IL_StrLChr** function applies the partial character string from the beginning of the source character sequence up to the first occurrence of the entered character "Char".
Return value: Address of target character string.

Interface description

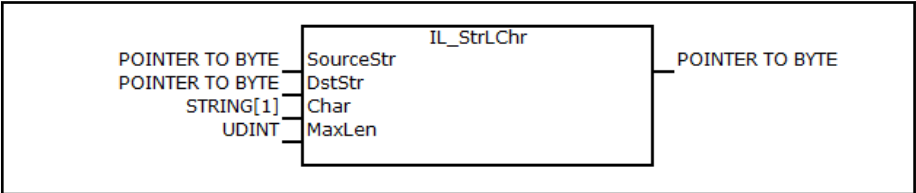


Fig. 4-122: FUN IL_StrLChr

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Char	STRING[1]	Delimiter of the partial character string
MaxLen	UDINT	Max. length of the element (in bytes/Char)

Tab. 4-113: FUN IL_StrLChr interface

Example 1

Str1: STRING := 'This is a string' ;

Str2: STRING;

IL_StrLChr (ADR(Str1), ADR(Str2), 'i', 100);

Return value: ADR(Str2)

Str2: 'Thi'

Example 2

Str1: STRING := 'This is a string' ;

IL_StrLChr (ADR(Str1), ADR(Str1), 'i', 100);

Return value: ADR(Str1)

Str1: 'Thi'

4.12.38 IL_StrRChr**Brief description**

The **IL_StrRChr** function applies the partial character string from last the occurrence of the entered character "Char" up to the end of the source character string.

Return value: Address of the target string.

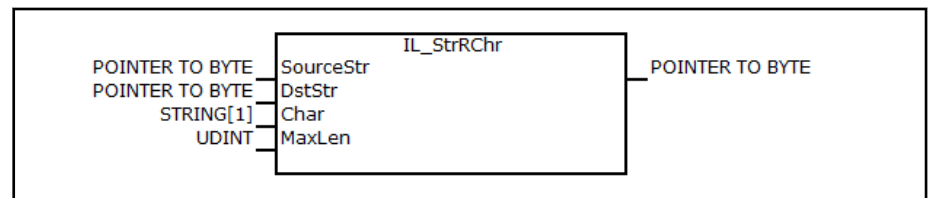
Interface description

Fig. 4-123: FUN IL_StrRChr

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Char	STRING[1]	Delimiter of the partial character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-114: FUN IL_StrRChr interface

Example 1

Str1: STRING := 'This is a string' ;

Str2: STRING;

IL_StrRChr (ADR(Str1), ADR(Str2), 'i', 100);

Return value: ADR(Str2)

Str2: 'ing'

Example 2

Str1: STRING := 'This is a string' ;

IL_StrRChr (ADR(Str1), ADR(Str1), 'i', 100);

Return value: ADR(Str1)

Str1: 'ing'

4.12.39 IL_StrSetElement

Brief description The source character string is split into elements by separator character strings.

 The **IL_StrSetElement** functions inserts the "Element" character string plus separator character string into the location specified by "ElementNum".

 Return value: Address of target character string.

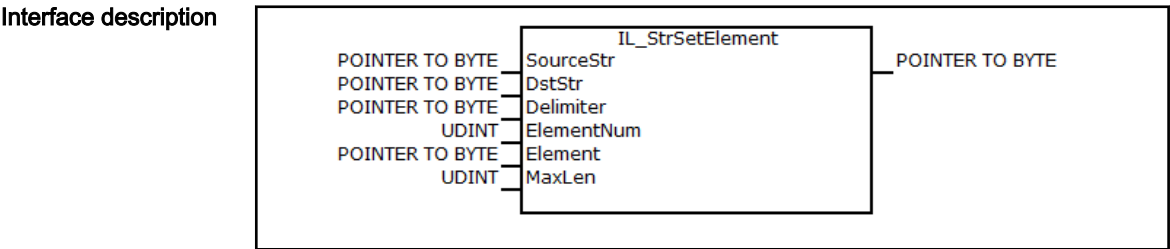


Fig. 4-124: FUN IL_StrSetElement

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Delimiter	POINTER TO BYTE	Address of the target character string
ElementNum	UDINT	Number of the element position the element is to be inserted into. The number of the first element is 1
Element	POINTER TO BYTE	Address of the element to be inserted
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-115: FUN IL_StrSetElement interface

Example 1

Str1: STRING := 'ab;;cd;;e';
Str2: STRING;
Str3: STRING := ';;' ;
Str4: STRING := 'fg';

IL_StrSetElement (ADR(Str1), ADR(Str2), ADR(Str3), 2 , ADR(Str4), 15);

Return value: ADR(Str2)
Str2: 'ab;;fg;;cd;;e'

Example 2

Str1: STRING := 'ab;;cd;;e';
Str2: STRING;
Str3: STRING := ';;' ;
Str4: STRING := 'fg';

IL_StrSetElement (ADR(Str1), ADR(Str2), ADR(Str3), 2 , ADR(Str4), 2);

Return value: ADR(Str2)
Str2: 'ab'

Example 3

```
Str1: STRING := 'ab;;cd;;e';
```

```
Str3: STRING := ';;';
```

```
Str4: STRING := 'fg';
```

```
IL_StrSetElement (ADR(Str1), ADR(Str1), ADR(Str3), 2, ADR(Str4), 5);
```

Return value: ADR(Str1)

Str1: 'ab;;fg;;cd;;e'

4.12.40 IL_StrSplit**Brief description**

The source character string is split into partial character strings by separators.

The address of each partial character string is stored in a specified array.

The data type of an array element is "POINTER TO STRING[255]"



The source characters string is changed when calling the function!

Interface description

Return value: Number of split partial character strings.

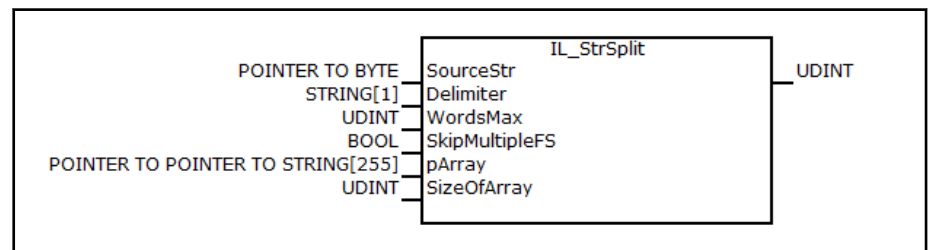


Fig. 4-125: FUN IL_StrSplit

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
Delimiter	STRING[1]	Separator
MaxWord	UDINT	Maximum number of partial character strings to be separated
SkipMultipleFS	BOOL	TRUE: Ignores multiple immediately following separators FALSE: If two or more separators follow each other immediately, only empty string "NUL" each is generated as partial character strings
pArray	POINTER TO POINTER TO STRING[255]	Address of the array containing the addresses of the partial character strings
SizeOfArray	UDINT	Length of the array (in bytes)

Tab. 4-116: FUN IL_StrSplit interface

Example 1

```
Str1: STRING := 'ab;;cd;;e';
```

```
arArray: ARRAY[0..200] OF POINTER TO STRING[255];
```

```
IL_StrSplit(ADR(Str), ';;', 4, FALSE, ADR(arArray), SIZEOF(arArray));
```

Return value: 4

arArray[1]: 'ab'

arArray[2]: "
arArray[3]: 'cd'
arArray [4]: 'e'

Example 2

Str1: STRING := 'ab;;cd;e';
arArray: ARRAY[0..200] OF POINTER TO STRING[255];
IL_StrSplit(ADR(Str), ';', 3, TRUE, ADR(arArray), SIZEOF(arArray));

Return value: 3
arArray[1]: 'ab'
arArray[2]: 'cd'
arArray[3]: 'e'

Example 3

Str1: STRING := 'ab;;cd;e';
arArray: ARRAY[0..200] OF POINTER TO STRING[255];
IL_StrSplit(ADR(Str), ';', 2, TRUE, ADR(arArray), SIZEOF(arArray));

Return value: 2
arArray[1]: 'ab'
arArray[2]: 'cd;e'

4.12.41 IL_StrTrimAll

Brief description The **IL_StrTrimAll** function removes all "Char" characters in one character string.

Return value: Address of target character string

Interface description

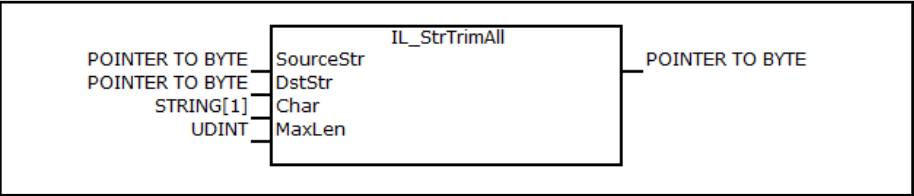


Fig. 4-126: FUN IL_StrTrimAll

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
Char	STRING[1]	Characters to be removed in the character string
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-117: FUN IL_StrTrimAll interface

Example 1

Str1: STRING := ';;;;;ab;;;;;cde;;;;;' ;

Str2: STRING;

IL_StrTrimAll (ADR(Str1), ADR(Str2), ',',10);

Return value: ADR(Str2)

Str2: 'abcde'

Example 2

Str1: STRING := ';;;;;ab;;;;;cde;;;;;' ;

Str2: STRING;

IL_StrTrimAll (ADR(Str1), ADR(Str2), ',',2);

Return value: ADR(Str2)

Str2: 'ab'

Example 3

Str1: STRING := ';;;;;ab;;;;;cde;;;;;' ;

IL_StrTrimAll (ADR(Str1), ADR(Str1), ',',10);

Return value: ADR(Str1)

Str1: 'abcde'

4.12.42 IL_StrTrimSpace**Brief description**

The **IL_StrTrimSpace** removes the spaces from a character string depending on the option specified.

Return value: Address of target character string

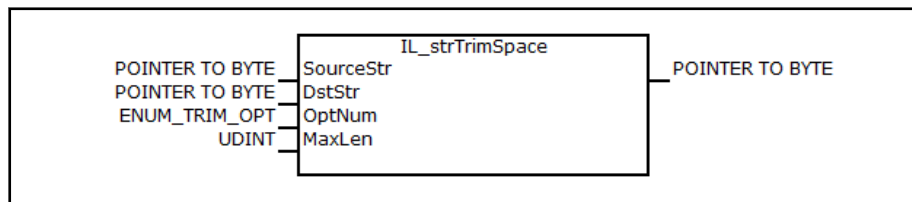
Interface description

Fig. 4-127: FUN IL_StrTrimSpace

Name	Type	Comment
SourceStr	POINTER TO BYTE	Address of source character string
DstStr	POINTER TO BYTE	Address of target character string
OptNum	ENUM_TRIM_OPT	Controls removal of spaces in the character string ENUM_TRIM_OPT, page 174
MaxLen	UDINT	Max. length of target character string (in bytes/Char)

Tab. 4-118: IL_StrTrimSpace interface

Example 1

Str1: STRING := 'This is a string' ;

Str2: STRING;

IL_StrTrimSpace (ADR(Str1), ADR(Str2), EN_TRIM_BOTH_ENDS, 100);

Return value: ADR(Str2)

Str2: 'This is a string'

Example 2

Str1: STRING := 'This is a string' ;
Str2: STRING;
IL_StrTrimSpace (ADR(Str1), ADR(Str2), EN_TRIM_BEGINNING, 100);

Return value: ADR(Str2)
Str2: 'This is a string'

Example 3

Str1: STRING := 'This is a string' ;
Str2: STRING;
IL_StrTrimSpace (ADR(Str1), ADR(Str2), EN_TRIM_END, 100);

Return value: ADR(Str2)
Str2: 'This is a string'

Example 4

Str1: STRING := 'This is a string' ;
Str2: STRING;
IL_StrTrimSpace (ADR(Str1), ADR(Str2), EN_TRIM_ALL, 100);

Return value: ADR(Str2)
Str2: 'This is a string'

4.12.43 ENUM_TRIM_OPT

The enumeration type **ENUM_TRIM_OPT** affects how spaces are removed from the character string.

Name	Value	Description
EN_TRIM_BOTH_ENDS	0	Removes all spaces at the beginning and the end of the character string
EN_TRIM_BEGINNING	1	Removes all spaces at the beginning of the character string
EN_TRIM_END	2	Removes all spaces at the end of the character string
EN_TRIM_ALL	3	Removes all spaces except for an individual space between two words

Tab. 4-119: ENUM_TRIM_OPT elements

4.13 Time and date calculation, version 2 (time)

4.13.1 Introduction and overview



Time base for all time functions and functions blocks is Unix time (since 1970-01-01). If an earlier date is to be output or converted, the date output will always be the basic date (1970-01-01).

The following functions and function blocks provide the system time and/or the date as a return value:

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Time (time and date calculation)	

IL_Date, page 175	Reads the current system date
IL_DateAndTime, page 175	Reads current system date and time
IL_ExtSysTimeDate, page 176	Reads current system date and time in the extended system format
IL_LocalTime, page 176	Converts control-internal UTC time to local time
IL_SysTime64, page 178	Reads current system date and time in microseconds since 01.01.1970
IL_SysTime64ToSysTimeDate, page 179	Converts the format of system date and system time
IL_SysTimeDate, page 179	Reads current system date and time
IL_SysTimeDateToSysTime64, page 180	Converts the format of system date and system time
IL_TimeOfDay, page 180	FUN reads the current system time

Tab. 4-120: RIL_Uilities, time and date calculation

4.13.2 IL_Date

Brief description

The function IL_Date reads the current date of the system.

Interface description

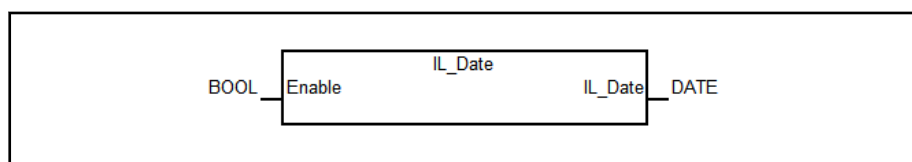


Fig. 4-128: Function IL_Date

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
RETURN	IL_Date	DATE	Current system date according to IEC61131-3

Tab. 4-121: Interface IL_Date

Functional description

The function IL_Date cyclically reads the system date formatted according to IEC61131-3 after processing enabled for the function via "Enable".

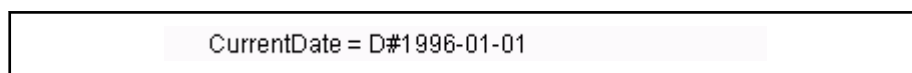


Fig. 4-129: Formatting IL_Date

Troubleshooting

No troubleshooting available for this function.

4.13.3 IL_DateAndTime

Brief description

The function IL_DateAndTime reads current system date and time.

Interface description

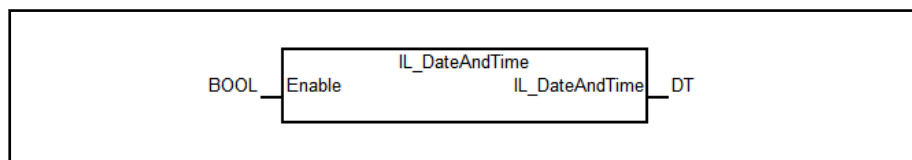


Fig. 4-130: Function IL_DateAndTime

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
RETURN	IL_DateAndTime	DT	Current system date and system time according to IEC61131-3

Tab. 4-122: Interface IL_DateAndTime

Functional description The function IL_DateAndTime cyclically reads the current system date as well as the current system time formatted according to IEC61131-3 after processing enabled for the function via "Enable".

CurrentDateAndTime = DT#1996-01-01-00:13:26

Fig. 4-131: Formatting IL_DateAndTime

Troubleshooting No troubleshooting available for this function.

4.13.4 IL_ExtSysTimeDate

Brief description The function function IL_ExtSysTimeDate reads current system date and time.

Interface description

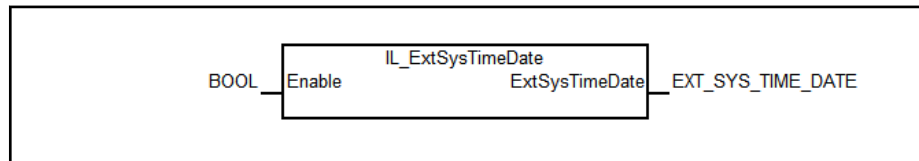


Fig. 4-132: IL_ExtSysTimeDate function block

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
VAR_OUTPUT	ExtSysTimeDate	EXT_SYS_TIME_DATE	Current system date and system time in the extended system format

Tab. 4-123: Interface IL_ExtSysTimeDate

Functional description The function block IL_ExtSysTimeDate cyclically reads the current system date as well as the current system time in the extended system format after processing enabled for the function block via "Enable".



The extended system format is composed of the system formats SYS_TIME64 and SYS_TIME_DATE.

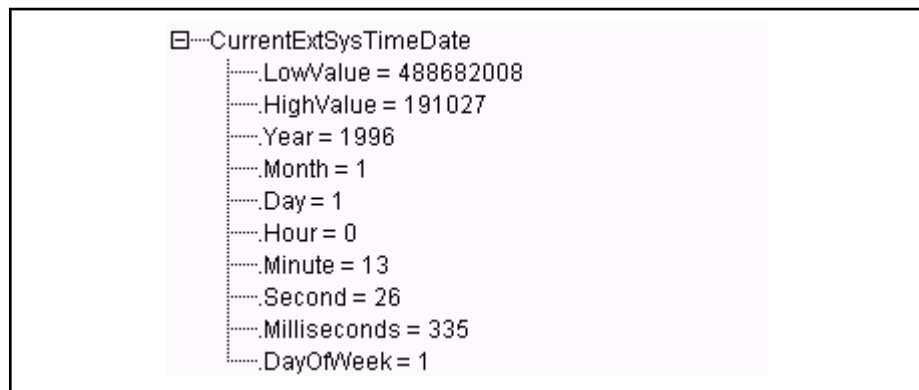


Fig. 4-133: Formatting IL_ExtSysTimeDate

Troubleshooting No error handling is available for this function block.

4.13.5 IL_LocalTime

Brief description The functionality of the "IL_LocalTime" function block is controlled via the "CurrentUTCTime" input.

If the "CurrentUTCTime" input contains the default value (input is not active), the function block reads out the system time of the control and converts it to local time using the time zone information.

If the "CurrentUTCTime" input does not contain the default value, the time value is converted to local time via the "CurrentUTCTime" and is output at the "LocalDateTime" output. In this case, the time is not read out of the control.

Interface description

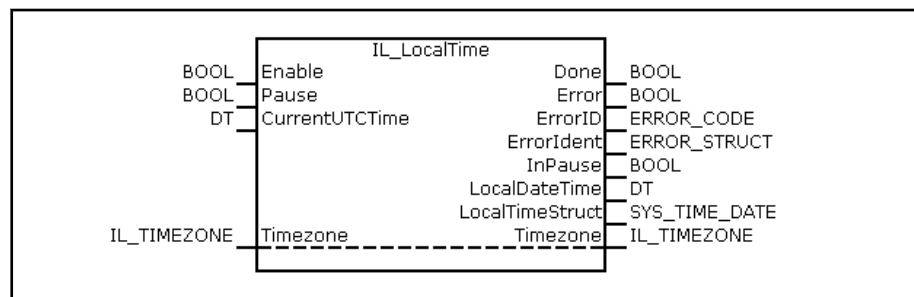


Fig. 4-134: IL_LocalTime function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Pause	BOOL	Pauses processing
	CurrentUTC-Time	DT	If a time value is specified via the input, this time is converted to local time and output at the "LocalDateTime" output. If the input is not active, the system time of the control is converted.
VAR_OUTPUT	Done	BOOL	Time updated; time outputs valid
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 178)
	InPause	BOOL	Processing paused
	LocalDateTime	DT	Local time as DT data type
	LocalTimeStruct	SYS_TIME_DATE	Local time as SYS_TIME_DATE
VAR_IN_OUT	Time zone	IL_TIMEZONE	Time zone description

Tab. 4-124: IL_LocalTime interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Rising edge at "Enable"
CurrentUTC-Time	DT			DT#1970-1-1-0:0:0	Continuous if Enable = TRUE
Time zone	IL_TIMEZONE				Rising edge at "Enable"

Tab. 4-125: Min./max. and default values of the IL_LocalTime function block inputs

Functional description

If the CurrentUTCTime input contains the default value (input is not active), the function block reads out the time zone information from the "Timezone" input in case of a rising edge at "Enable". The system time is cyclically read

and converted to local time using the time zone information and the result is set on the outputs "LocalDateTime" and "LocalTimeStruct". The output is refreshed each second. At each refresh, the "Done" output is set for one PLC cycle.

If the CurrentUTCTime input does not contain the default value, the time value is read in via the nCurrentUTCTime input in case of a set "Enable" input. This time value is converted to local time and output at the "LocalDateTime" output. In this case, the time is not read out of the control.

The "Pause" input pauses the processing. If this input is set, no time is read and the outputs "LocalDateTime" and "LocalTimeStruct" are frozen.

If an error occurs while processing, the "Error" output is set. "ErrorID" and "ErrorIdent" contain detailed information on the error.

Troubleshooting The function block uses the F_RELATED_TABLE, 16#0170 error table. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00002500	16#00000000	Winter time settings inconsistent
INPUT_RANGE_ERROR	16#00002500	16#00000001	Time in WinterDiffUtc bigger than 24 hours
INPUT_RANGE_ERROR	16#00002500	16#00000002	Value in SwiWinterMonth higher than 12
INPUT_RANGE_ERROR	16#00002500	16#00000003	Value in SwiWinterWeek higher than 5
INPUT_RANGE_ERROR	16#00002500	16#00000004	Value in SwiWinterWeekday higher than 7
INPUT_RANGE_ERROR	16#00002500	16#00000005	Value in SwiWinterDayYear higher than 365
INPUT_INVALID_ERROR	16#00002501	16#00000000	Summer time settings inconsistent
INPUT_RANGE_ERROR	16#00002501	16#00000001	Time in SummerDiffUtc bigger than 24 hours
INPUT_RANGE_ERROR	16#00002501	16#00000002	Value in SwiSummerMonth higher than 12
INPUT_RANGE_ERROR	16#00002501	16#00000003	Value in SwiSummerWeek higher than 5
INPUT_RANGE_ERROR	16#00002501	16#00000004	Value in SwiSummerWeekday higher than 7
INPUT_RANGE_ERROR	16#00002501	16#00000005	Value in SwiSummerDayYear higher than 365
INPUT_INVALID_ERROR	16#00002502	16#00000000	Error at time conversion due to incorrect time zone information

Tab. 4-126: Error codes generated by IL_LocalTime

4.13.6 IL_SysTime64

Brief description The function function IL_SysTime64 reads current system date and time.

Interface description

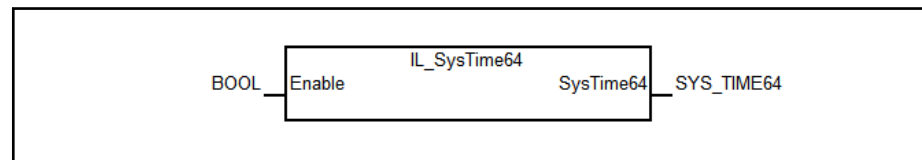


Fig. 4-135: IL_SysTime64 function block

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
VAR_OUTPUT	SysTime64	SYS_TIME64	Current system date and system time in microseconds since 01.01.1970

Tab. 4-127: Interface IL_SysTime64

Functional description The function block IL_SysTime64 reads the current system date as well as the current system time in microseconds since 01.01.1970 after processing enabled for the function via "Enable".

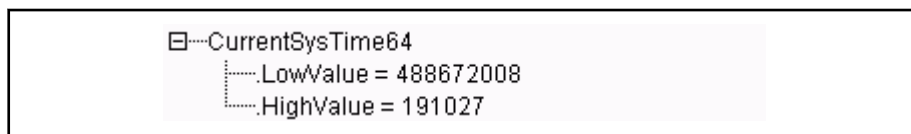


Fig. 4-136: Formatting IL_SysTime64

Troubleshooting No error handling is available for this function block.

4.13.7 IL_SysTime64ToSysTimeDate

Brief description The function block IL_SysTime64ToSysTimeDate serves for the conversion of the system date and the system time.

Interface description

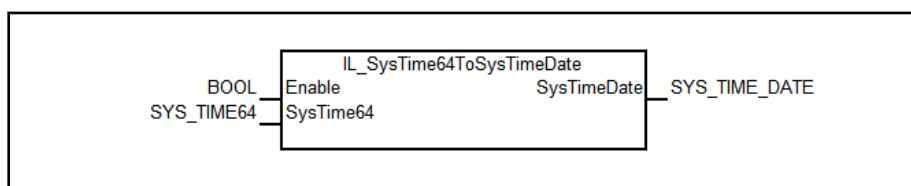


Fig. 4-137: IL_SysTime64ToSysTimeDate function block

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing enabled for function block (cyclic, state-controlled)
	SysTime64	SYS_TIME64	System date and system time in microseconds since 01.01.1970
VAR_OUTPUT	SysTimeDate	SYS_TIME_DATE	System date and system time in the system format

Tab. 4-128: Interface IL_SysTime64ToSysTimeDate

Functional description After processing enabled for the function block, the function block IL_SysTime64ToSysTimeDate uses "Enable" to convert the system date including the system time available at the input "SysTime64" in microseconds to the system date and the system time in the system format (basis: 01.01.1970).

Troubleshooting No error handling is available for this function block.

4.13.8 IL_SysTimeDate

Brief description The function function IL_SysTimeDate reads current system date and time.

Interface description

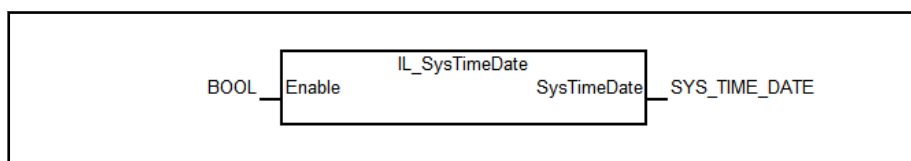


Fig. 4-138: IL_SysTimeDate function block

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
VAR_OUTPUT	SysTimeDate	SYS_TIME_DATE	Current system date and system time in the system format

Tab. 4-129: Interface IL_SysTimeDate

Functional description The function block IL_SysTimeDate cyclically reads the current system date as well as the current system time in the system format after processing enabled for the function block via "Enable".

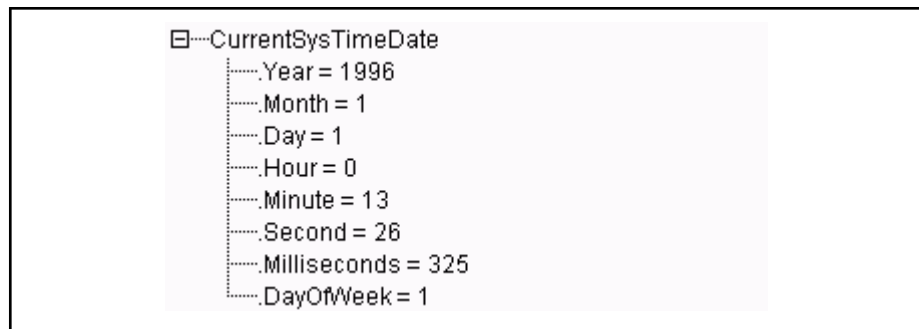


Fig. 4-139: Formatting IL_SysTimeDate

Troubleshooting No error handling is available for this function block.

4.13.9 IL_SysTimeDateToSysTime64

Brief description The function block IL_SysTimeDateToSysTime64 serves for the format conversion of the system date and the system time.

Interface description

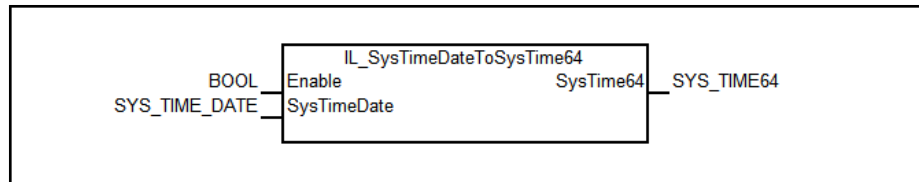


Fig. 4-140: IL_SysTimeDateToSysTime64 function block

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing enabled for function block (cyclic, state-controlled)
	SysTimeDate	SYS_TIME_DATE	System date and system time in the system format
VAR_OUTPUT	SysTime64	SYS_TIME64	System date and system time in microseconds since 01.01.1970

Tab. 4-130: Interface IL_SysTimeDateToSysTime64

Functional description After processing enabled for the function block, the function block IL_SysTimeDateToSysTime64 uses "Enable" to cyclically convert the system date including the system time available at the input "SysTimeDate" in microseconds (basis: 01.01.1970).

Troubleshooting No error handling is available for this function block.

4.13.10 IL_TimeOfDay

Brief description The function IL_TimeOfDay reads the current time of the system.

Interface description

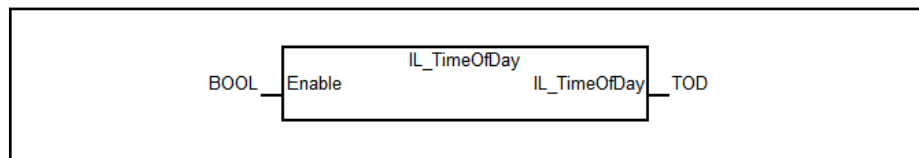


Fig. 4-141: Function IL_TimeOfDay

	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function enabled (cyclic, state-controlled)
RETURN	IL_TimeOfDay	TOD	Current system time according to IEC61131-3

Tab. 4-131: Interface IL_TimeOfDay

Functional description The function IL_TimeOfDay cyclically reads the system time formatted according to IEC61131-3 after processing enabled for the function via "Enable".

CurrentTimeOfDay = TOD#00:13:26.335

Fig. 4-142: Formatting IL_TimeOfDay

Troubleshooting

No troubleshooting available for this function.

4.13.11 IL_TIMEZONE

The data structure contains information on the time zone.

Element	Data type	Min. value	Max. value	Default value	Description
WinterDiffUtc	TIME			t#0s	Difference from local winter time to UTC
WinterLoc	BOOL			FALSE	TRUE: East from Greenwich FALSE: West from Greenwich
SwiWinterMonth	USINT	0	12	0	Month when converting to winter time 0: Not used 1..12: month
SwiWinterWeek	USINT	0	5	0	Week in the month of winter time conversion 0: Not used 1..4 week of the conversion 5: Last week in the month
SwiWinterWeekday	USINT	0	7	0	Weekday when converting to winter time 0: Not used 1..7: Weekday (1: Mon, 7: Sun)
SwiWinterDayYear	UINT	0	365	0	Day in the year of the conversion to winter time 0: Not used
SwiWinterTime	TOD	t#0s	t#23h59m59s	t#0s	Time when converting to winter time
SummerDiffUtc	TIME			t#0s	Difference from local summer time to UTC
SummerLoc	BOOL			FALSE	TRUE: East from Greenwich FALSE: West from Greenwich
SwiSummerMonth	USINT	0	12	0	Month when converting to summer time 0: Not used 1..12: month
SwiSummerWeek	USINT	0	5	0	Week in the month of summer time conversion 0: Not used 1..4 week of the conversion 5: Last week in the month

Element	Data type	Min. value	Max. value	Default value	Description
SwiSummerWeekday	USINT	0	7	0	Weekday when converting to summer time 0: Not used 1..7: Weekday (1: Mon, 7: Sun)
SwiSummerDayYear	UINT	0	365	0	Day in the year of the conversion to summer time 0: Not used
SwiSummerTime	TOD	t#0s	t#23h59m59s	t#0s	Time when converting to summer time
South	BOOL			FALSE	Location south of the equator

Tab. 4-132: IL_TIMEZONE data structure

Predefined time zones

To facilitate the time zone handling, several important time zones are already predefined. This, the structure has to be filled out only in exceptional cases.

If predefined time zones are used, the "Timezone." prefix has to be used.

Timezone.CEST for "Central European time zone"

Constant	Time zone	Description
ACST	ACST	UTC+9:30 Australia
ACDT	ACST/ ACDT	UTC+9:30/UTC+10:30 Australia
AEST	AEST	UTC+10 Australia
AEDT	AEST/ AEDT	UTC+10/UTC+11 Australia
AMT	AMT, BOT	UTC-4 Brazil, Bolivia
AMST	AMT/ AMST	UTC-4/UTC-3 Brazil
ART	ART	UTC-3 Argentina
ARST	ART/ARST	UTC-3/UTC-2 Argentina
AWST	AWST	UTC+8 Australia
BRT	BRT	UTC-3 Brazil
BRST	BRT/BRST	UTC-3/UTC-2 Brazil

Constant	Time zone	Description
BST	GMT/BST	UTC+0/UTC+1 Great Britain
CAT	CAT	UTC+2 Botswana, Burundi, Congo, Mozambique, Rwanda, Zambia, Zimbabwe
CET	CET	UTC+1 Algeria, Tunisia
CEST	CET/CEST	UTC+1/UTC+2 Albania, Andorra, Austria, Belgium, Bosnia-Herzegovina, Croatia, Czech Republic, Denmark, France, Germany, Gibraltar, Vatican City, Hungary, Italy, Kosovo, Lichtenstein, Luxemburg, Macedonia, Malta, Monaco, Montenegro, Netherlands, Norway, Poland, San Marino, Serbia, Slovakia, Slovenia, Spain - except Canary Islands, Sweden, Switzerland
CCT	CCT, HKT	UTC+8 China, Macau, Taiwan, Hong Kong
CDT	CST/CDT	UTC-6/-5 Canada, USA, Mexico
CVT	CVT	UTC-1 Cape Verde
EAT	EAT	UTC+3 Djibouti, Eritrea, Ethiopia, Kenya, Madagascar, Somalia, Sudan, Tanzania, Uganda
EDT	EST/EDT	UTC-5/UTC-4 Canada, USA
EET	EET	UTC+2 Libya
EEST	EET/EEST	UTC+2/UTC+3 Belarus, Bulgaria, Estonia, Finland, Greece, Latvia, Lithuania, Moldova, Romania, Russia (Kaliningrad), Turkey, Ukraine, Egypt
FNT	FNT	UTC-2 Brazil
GMT	GMT	UTC+0 Iceland, Burkina Faso, Ivory Coast, Gambia, Ghana, Guinea-Bissau, Guinea, Liberia, Mali, Mauritania, Saint Helena, Senegal, Sierra Leone, Togo
ICT	ICT	UTC+7 Cambodia, Laos, Thailand, Vietnam
IST	IST	UTC+5:30 India

Constant	Time zone	Description
JST	JST	UTC+9 Japan
MSK	MSK/MSD	UTC+3/UTC+4 Russia
MDT	MST/MDT	UTC-7/UTC-6 USA
PDT	PDT	UTC-8/UTC-7 Canada, USA
SAST	SAST	UTC+2 Lesotho, South Africa, Swaziland
WAT	WAT	UTC+1 Angola, Benin, Central African Republic, Chad, Congo, Equatorial Guinea, Gabon, Niger, Nigeria
WAST	WAT/ WAST	UTC+1/UTC+2 Namibia
WEST	WET/ WEST	UTC+0/UTC+1 Faroe Islands, Portugal, Canary Islands, Madeira, Morocco

Tab. 4-133: Predefined time zones

4.14 Double-linked lists (Utilities/Logic)

4.14.1 Introduction and overview

The following function block was designed for applications at which a double-linked list is required. Elements can be added and deleted.

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Double-linked lists (Utilities/Logic)	
IL_LinkedList , page 184,	Manages double-linked lists (used for ring buffer, FIFOs and LIFOs) <i>The function block uses the data types</i> IL_LINKED_LIST_MAINTENANCE, page 188 IL_LINKED_LIST_MODE, page 189

Tab. 4-134: RIL_Uilities, double-linked lists

4.14.2 IL_LinkedList

Brief description

The function block maintains a double-linked list. The list content can be specified by the user.



In 14VRS, the "IL_LinkedList (ML_TechBase library)" was moved to the "RIL_Uilities library".

Interface description

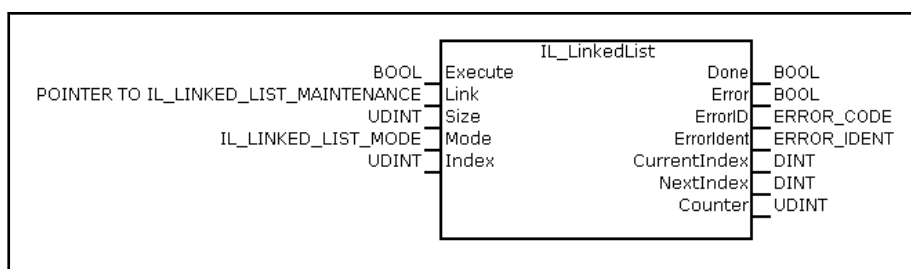


Fig. 4-143: IL_LinkedList function block

I/O type	Name	Data type	Description
VAR_INPUT	Execute	BOOL	The function block starts its functionality at a rising edge
	Link	POINTER TO IL_LINKED_LIST_MAINTENANCE	Pointer to an array of IL_LINKED_LIST_MAINTENANCE structures required to maintain the list IL_LINKED_LIST_MAINTENANCE, page 188
	Size	UDINT	Number of list elements to be maintained
	Mode	IL_LINKED_LIST_MODE	Mode that executes the function block, e.g. adding or removing an element IL_LINKED_LIST_MODE, page 189
	Index	DINT	Index on which the operation is to be applied
VAR_OUTPUT	Done	BOOL	If this output is TRUE, the function block has successfully completed its operation
	Error	BOOL	Indicates that an error occurred in the function block. The output data is invalid
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	Error structure with further subdivision of error classes Error codes, page 188
	CurrentIndex	DINT	Current index
	NextIndex	DINT	Next index
	Counter	UDINT	Number of elements in the list

Tab. 4-135: Interface variables of the IL_LinkedList function block

Min./max. and default values of the inputs

The following value ranges and default values can be applied at the function block:

Name	Type	Min. value	Max. value	Default value
Link	0	0	16#FFFFFFFF	0
Size	UDINT	0	16#FFFFFFFF	0
Mode	IL_LINKED_LIST_MODE	ModeInitClear	ModeCheckIndex	ModeInitClear
Index	UDINT	0	16#FFFFFFFF	0

Tab. 4-136: Min./max. and default values of the IL_LinkedList function block

Functional description

The function block maintains a double-linked list. This list is stored in an array in which only the indices have to be maintained. To maintain the list, create an array of type IL_LINKED_LIST_MAINTENANCE (see [Using the IL_LINKED_LIST_MAINTENANCE structure, page 186](#)) that has to be as large as its own data.

Using the IL_LINKED_LIST_MAINTENANCE structure

The list is manipulated by calling the function block. The manipulation type can be defined by the "Mode" input ([IL_LINKED_LIST_MODE](#), [page 189](#)).

The data structure `IL_LINKED_LIST_MAINTENANCE` is used by the function block and may not be overwritten. Since users can determine the length of the linked list, they have to create the maintenance structures for the list as well.

Therefore, define an array of the type `IL_LINKED_LIST_MAINTENANCE`, [page 188](#).

The structure can be used for debugging.

"Prev" is a pointer to the preceeding list element

"Next" is a pointer to the following list element.

If the value is 0, it is the end or the beginning of the array or an unused element.

The inputs "Link" and "Size" of the function block may not be modified after initialized by "Mode ModelInitClear". This also applies when changing online e.g. if the length of the data structure is changed.

Examples Example of the function block use:

Defining variables:

```
(* Maximum array size *)
uiMAX_ARRAY_SIZE: INT := 10;
(* My data structure array *)
arMyData: ARRAY [1..uiMAX_ARRAY_SIZE] OF MY_STRUCTURE;
(* List management array*)
arLinkedList: ARRAY [1..uiMAX_ARRAY_SIZE] OF IL_LINKED_LIST_MAINTENANCE;
(* Function block instance *)
fbMyList: IL_LinkedList;
(* Index variable*)
udiIndex: DINT;
```

Call for initialization:

```
fbMyList(Execute:= TRUE,
  Link:= ADR(arLinkedList),
  Size:= uiMAX_ARRAY_SIZE,
  Mode:= ModeInitClear,
  Index:= ,
  Done=> ,
  Error=> ,
  ErrorId =>,
  ErrorIdent=> ,
  CurrentIndex=> ,
  NextIndex=>);
```

Adding a head to the list:

```
(* Reset "Execute" input first (value of mode is not important) *)
fbMyList(Execute:= FALSE,
  Link:= ADR(arLinkedList),
  Size:= uiMAX_ARRAY_SIZE,
  Mode:= ModeInitClear,
  Index:= ,
  Done=> ,
  Error=> ,
  ErrorId =>,
  ErrorIdent=> ,
  CurrentIndex=> ,
  NextIndex=>);

(* Add head *)
fbMyList(Execute:= TRUE,
  Link:= ADR(arLinkedList),
  Size:= uiMAX_ARRAY_SIZE,
  Mode:= ModeAddHead,
  Index:= ,
```

```

        Done=> ,
        Error=> ,
        ErrorId =>,
        ErrorIdent=> ,
        CurrentIndex=> udiIndex,
        NextIndex=>);

(* Return value in CurrentIndex contains index for list head or
-1 if an error occurred. *)
IF udiIndex <> -1 THEN
    arMyData[udiIndex].StructureMember := MyValue;
END_IF;

```

Scroll through a list:

```

(* Reset "Execute" input first (value of mode is not important) *)
fbMyList(Execute:= FALSE,
    Link:= ADR(arLinkedList),
    Size:= uiMAX_ARRAY_SIZE,
    Mode:= ModeGetHead,
    Index:= ,
    Done=> ,
    Error=> ,
    ErrorId =>,
    ErrorIdent=> ,
    CurrentIndex=> ,
    NextIndex=>);

(* Determine head *)
fbMyList(Execute:= TRUE,
    Link:= ADR(arLinkedList),
    Size:= uiMAX_ARRAY_SIZE,
    Mode:= ModeGetHead,
    Index:= ,
    Done=> ,
    Error=> ,
    ErrorId =>,
    ErrorIdent=> ,
    CurrentIndex=> udiIndex,
    NextIndex=>);

(* Check return value (-1 means error) *)
WHILE (udiIndex > 0) DO
    (* Return value in CurrentIndex contains index in list *)
    arMyData[udiIndex].StructureMember := MyValue;

    (* Reset "Execute" input again *)
    fbMyList(Execute:= FALSE,
        Link:= ADR(arLinkedList),
        Size:= uiMAX_ARRAY_SIZE,
        Mode:= ModeGetNext,
        Index:= ,
        Done=> ,
        Error=> ,
        ErrorId =>,
        ErrorIdent=> ,
        CurrentIndex=> ,
        NextIndex=>);

    (* Get next element*)
    fbMyList(Execute:= TRUE,
        Link:= ADR(arLinkedList),
        Size:= uiMAX_ARRAY_SIZE,
        Mode:= ModeGetNext,
        Index:= ,
        Done=> ,
        Error=> ,
        ErrorId =>,
        ErrorIdent=> ,
        CurrentIndex=> udiIndex,
        NextIndex=>);
END_WHILE

```

Output assignment depending on the "Mode" input

Depending on the "Mode" input assignment, the "Index" input and the "CurrentIndex" and "NextIndex" are configured accordingly. The following table lists the respective values. An X stands for a value that is unchanged with respect to the previous call. If an error occurs, -1 is returned for "CurrentIndex".

Mode	Index	CurrentIndex	NextIndex
ModeInitClear	X	-1	Next free index
ModeAddTail	X	Tail index	Next free index
ModeAddHead	X	Head index	Next free index
ModeRemoveTail	X	New tail index	Next free index
ModeRemoveHead	X	New head index	Next free index
ModeRemoveIndex	Index for removal	Previous element index	Next free index
ModeGetHead	X	Head index	X
ModeGetTail	X	Tail index	X
ModeGetNext	Previous index	Subsequent element index	X
ModeGetPrev	Next index	Previous element index	X
ModeGetEmptyIndex	X	X	Next free index
ModeCheckIndex	Testing index	1: Found -1: Not found	-1

Tab. 4-137: Function block outputs with respect to "Mode"

Error codes The function block uses the error table **F_RELATED_TABLE**. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#1900	16#0	The "Index" input is not in the valid range
INPUT_INVALID_ERROR	16#1901	16#0	"Link" input is a zero pointer
ACCESS_ERROR	16#1902	16#xxx	An index is still linked in the list although it is empty (internal error)
ACCESS_ERROR	16#1903	16#0	An element in the list is to be cleared, but the list is already empty
RESOURCE_ERROR	16#1904	16#0	List is full
STATE_MACHINE_ERROR	16#1905	16#0	"Mode" input has an invalid value
RESOURCE_ERROR	16#1906	16#0	"Size" input changed again after initialization (e.g. by online change).

Tab. 4-138: Error codes of the IL_LinkedList function block

4.14.3 IL_LINKED_LIST_MAINTENANCE

The structure is used by the function block [IL_LinkedList](#), page 184,.

"Prev" is a pointer for the previous list element, "Next" for the subsequent element. If the value is 0, it is the end or the beginning of the array or an unused element.

Due to this structure, debugging lists is facilitated.

Type definition

```

TYPE IL_LINKED_LIST_MAINTENANCE :
STRUCT
    Prev: POINTER TO DINT;
    Next: POINTER TO DINT;
END_STRUCT
END_TYPE

```

4.14.4 IL_LINKED_LIST_MODE

Use the enumeration type to define the action to be executed.

The enumeration is used by the function block [IL_LinkedList](#), page 184.

Enumeration	Significance
ModeInitClear	The linked list is initialized. All entries are deleted
ModeAddTail	An element is added at the tail of the list. If the list is empty, the first element is created
ModeAddHead	An element is added at the head of the list. If the list is empty, the first element is created
ModeRemoveTail	The last element in the list is removed
ModeRemoveHead	The first element in the list is removed
ModeRemoveIndex	The element in the index "Index" is removed
ModeGetHead	Returns the index of the head of the list
ModeGetTail	Returns the index of the tail of list
ModeGetNext	Returns the subsequent element in the list
ModeGetPrev	Returns the previous element in the list
ModeGetEmptyIndex	Returns the next free index in the list
ModeCheckIndex	Checks whether the specified index is used in the linked list. The result is returned in the "CurrentIndex": 1: Index is used -1: Index is not used

Tab. 4-139: Enumerations and their meaning in the function block *IL_LinkedList*

4.15 Modulo value calculation (Utilities/Logic)

4.15.1 Introduction and overview

The following functions calculate a modulo value. Depending on the selected function, different value ranges are used.



The mentioned functions changed their names in 14VRS from MB_* to IL_* and moved from "ML_TechBase" to "RIL_Uilities".

Areas of application and functions/function blocks of RIL_Uilities

Area of application (folder in library)	
Function blocks	Description
Modulo value calculation (Utilities/Logic)	
IL_GetModuloType01 , page 190	Calculates a modulo value. Value range between 0.0 and + modulo value (REAL)
IL_GetModuloType02 , page 191	Calculates a modulo value. Value range between 0 and + modulo value (DINT)
IL_GetModuloType03 , page 192	Calculates a modulo value. Value range between - modulo value and + modulo value (REAL)
IL_GetModuloType04 , page 193	Calculates a modulo value. Value range between - modulo/2 and + modulo value/2 (REAL)

Tab. 4-140: *RIL_Uilities, Modulo value calculation*

4.15.2 IL_GetModuloType01

Brief description The function returns the modulo value. The value range returned is between 0.0 and + modulo value.



In 14VRS, The "MB_GetModuloType01 (ML_TechBase library)" function was renamed to "IL_GetModuloType01".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities
MLD with MPx18	MX_Technology07.lib

Tab. 4-141: Reference table of the IL_GetModuloType01

Interface description

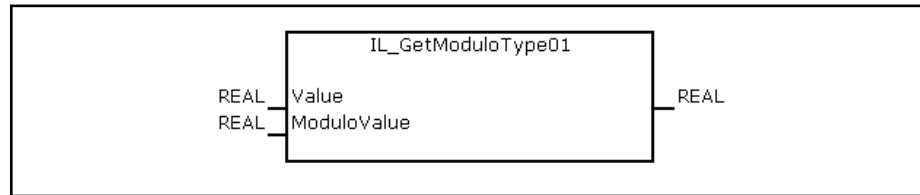


Fig. 4-144: IL_GetModuloType01 function

I/O type	Name	Data type	Description
VAR_INPUT	Value	REAL	Value at which the modulo function is to be applied
	ModuloValue	REAL	Modulo value to be used for the modulo operation
RETURN		REAL	Modulo value: The value range returned is between 0.0 and + modulo value.

Tab. 4-142: Interface variables of the IL_GetModuloType01 function

Min./max. and default values The following table lists the min./max. and default values of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Value	REAL	- 1.175494e-38	3.402823e+38	0.0	With every call
ModuloValue	REAL	- 1.175494e-38	3.402823e+38	360.0	With every call

Tab. 4-143: Min./max. and default values of the IL_GetModuloType01 function

Functional description

The function returns the modulo value. The value range returned is between 0.0 and + modulo value. The positive and the negative limit value are not part of the value range (also see the following diagram which shows the value transferred to the function on the x-coordinates and the value returned from the function on the y-coordinates).

If 0.0 is transferred for "ModuloValue", the function returns the value "Value".

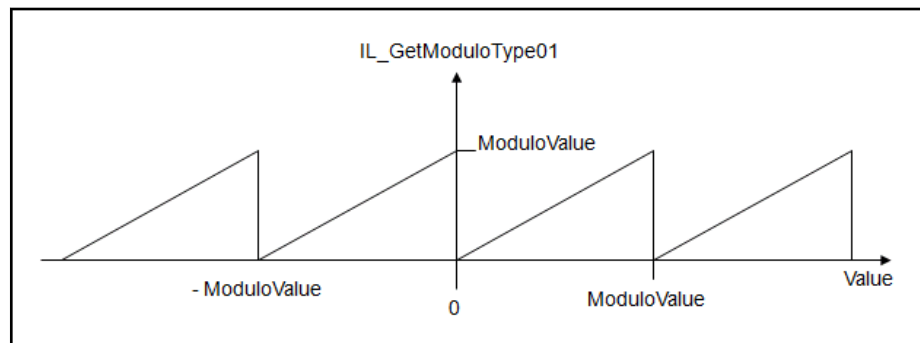


Fig. 4-145: Representation of the modulo function

Troubleshooting The function does not generate any errors.

4.15.3 IL_GetModuloType02

Brief description The function returns the modulo value. The value range returned is between 0 and + modulo value



In 14VRS, The "MB_GetModuloType02 (ML_TechBase library)" function was renamed to "IL_GetModuloType02".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities
MLD with MPx07 without MA function package	MX_Technology_07.lib

Tab. 4-144: Reference table of the IL_GetModuloType02

Interface description

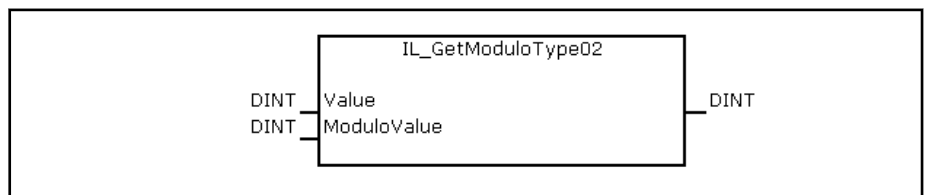


Fig. 4-146: IL_GetModuloType02 function

I/O type	Name	Data type	Description
VAR_INPUT	Value	DINT	Value at which the modulo function is to be applied
	ModuloValue	DINT	Modulo value to be used for the modulo operation
RETURN		DINT	Modulo value: The value range returned is between 0 and + modulo value

Tab. 4-145: Interface variables of the IL_GetModuloType02 function

Min./max. and default values The following table lists the min./max. and default values of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Value	DINT	- 2147483648	2147483647	0	With every call
ModuloValue	DINT	- 2147483648	2147483647	360	With every call

Tab. 4-146: Min./max. and default values of the IL_GetModuloType02 function

Functional description

The function returns the modulo value. The value range returned is between 0 and + modulo value. The positive and the negative limit value are not part of the value range (also see the following diagram which shows the value transferred to the function on the x-coordinates and the value returned from the function on the y-coordinates).

If 0 is transferred for "ModuloValue", the function returns the value "Value".

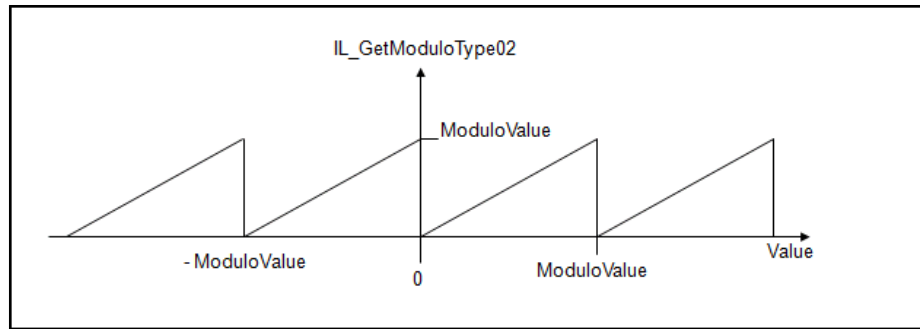


Fig. 4-147: Representation of the modulo function

Troubleshooting The function does not generate any errors.

4.15.4 IL_GetModuloType03

Brief description The function returns the modulo value. The returned value range is between - modulo value and + modulo value.



In 14VRS, The "MB_GetModuloType03 (ML_TechBase library)" function was renamed to "IL_GetModuloType03".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities
MLD with MPx18	MX_Technology07.lib

Tab. 4-147: Reference table of the IL_GetModuloType03

Interface description

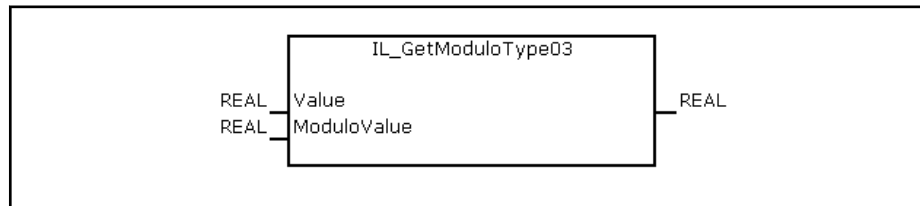


Fig. 4-148: IL_GetModuloType03 function

I/O type	Name	Data type	Description
VAR_INPUT	Value	REAL	Value at which the modulo function is to be applied
	ModuloValue	REAL	Modulo value to be used for the modulo operation
RETURN		REAL	Modulo value: The returned value range is between - modulo value and + modulo value.

Tab. 4-148: Interface variables of the IL_GetModuloType03 function

Min./max. and default values The following table lists the min./max. and default values of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Value	REAL	- 1.175494e-38	3.402823e+38	0.0	With every call
ModuloValue	REAL	- 1.175494e-38	3.402823e+38	360.0	With every call

Tab. 4-149: Min./max. and default values of the IL_GetModuloType03 function

Functional description The function returns the modulo value. The returned value range is between - modulo value and + modulo value. The positive and the negative limit value are not part of the value range (also see the following diagram which shows

the value transferred to the function on the x-coordinates and the value returned from the function on the y-coordinates).

If 0.0 is transferred for "ModuloValue", the function returns the value "Value".

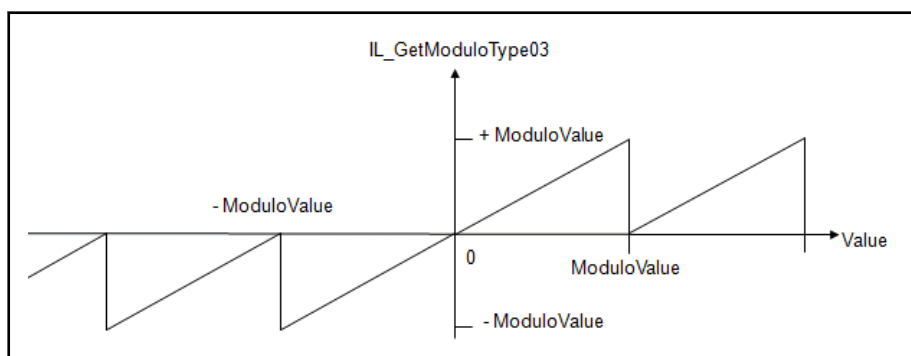


Fig. 4-149: Representation of the modulo function

Troubleshooting The function does not generate any errors.

4.15.5 IL_GetModuloType04

Brief description The function returns the modulo value. The returned value range is between - modulo / 2 and + modulo / 2.



In 14VRS, The "MB_GetModuloType04 (ML_TechBase library)" function was renamed to "IL_GetModuloType04".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_Uilities
MLD with MPx18	MX_Technology07.lib

Tab. 4-150: Reference table of the IL_GetModuloType04

Interface description

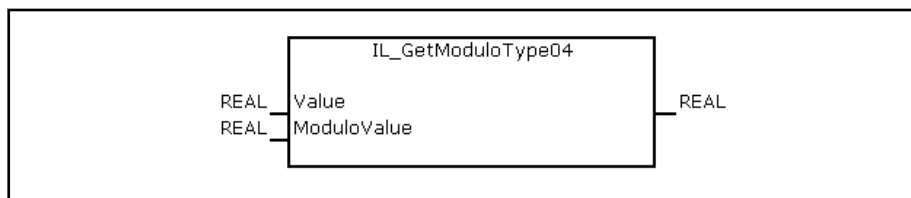


Fig. 4-150: IL_GetModuloType04 function

I/O type	Name	Data type	Description
VAR_INPUT	Value	REAL	Value at which the modulo function is to be applied
	ModuloValue	REAL	Modulo value to be used for the modulo operation
RETURN		REAL	Modulo value: The returned value range is between - modulo / 2 and + modulo / 2.

Tab. 4-151: Interface variables of the IL_GetModuloType04 function

Min./max. and default values The following table lists the min./max. and default values of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Value	REAL	- 1.175494e-38	3.402823e+38	0.0	With every call
ModuloValue	REAL	- 1.175494e-38	3.402823e+38	360.0	With every call

Tab. 4-152: Min./max. and default values of the IL_GetModuloType04 function

Functional description The function returns the modulo value. The returned value range is between $-\text{modulo} * 0.5$ and $+\text{modulo} * 0.5$. The positive and the negative limit value are part of the value range (also see the following diagram which shows the value transferred to the function on the x-coordinates and the value returned from the function on the y-coordinates).

If 0.0 is transferred for "ModuloValue", the function returns the value "Value".

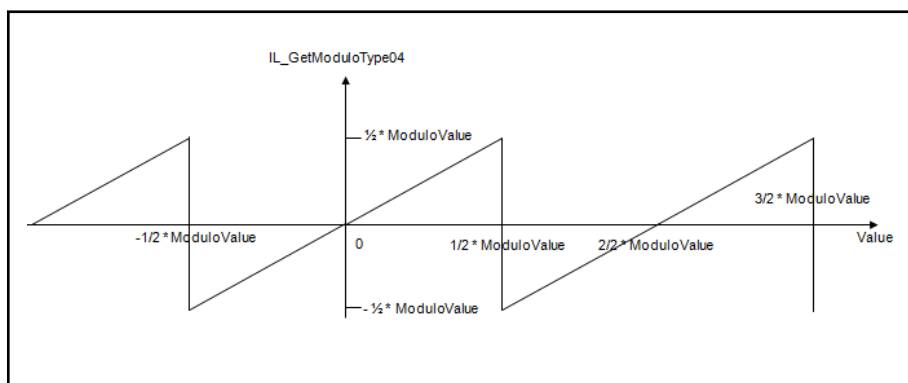


Fig. 4-151: Representation of the modulo function

Troubleshooting The function does not generate any errors.

5 RIL_ModbusTCP.library

5.1 Introduction and overview



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

The Modbus TCP exchanges data between Ethernet devices supporting the Modbus protocol. This documentation focuses on HMI devices. The Modbus slave (server) is the control used for the Modbus masters (clients) to connect. A Modbus TCP server is implemented in each of the [IL_ModbusTCPServer](#), page 209, and [IL_ModbusTCPServerType02](#), page 212, function blocks in the RIL_ModbusTCP.library.

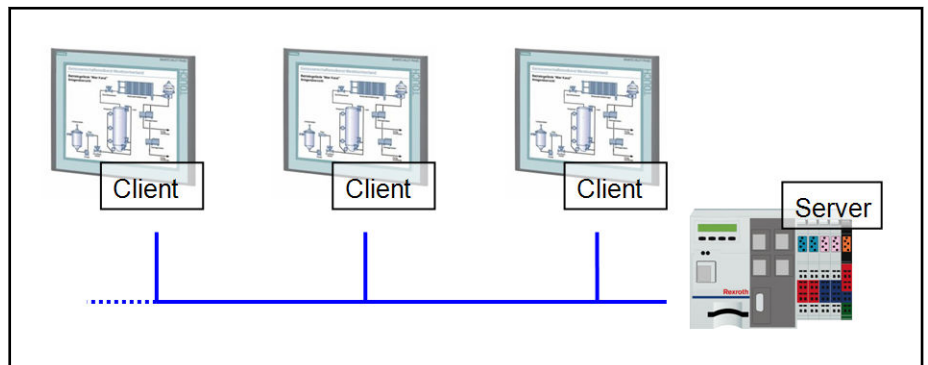


Fig. 5-1: Connecting several Modbus clients to a Modbus server



The functions blocks "IL_ModBusTCPServer" and "IL_ModBusTCPServerType02" differ in the following:

In contrast to the Modbus specification, the data ranges for "Coil Data" and "Discrete Input Data" are summarized under "BitData", the "Holding Register" and "Input Register" are summarized under "RegisterData" in the "IL_ModBusTCPServer" function block.

It is recommended to use the function block [IL_ModbusTCPServerType02](#), page 212,.

Target system	The RIL_ModBusTCP library is supported by the MLC system from version 10VRS.
ModbusTCP, terms	• Coils: Coils form a bit array with a length of 65536 bits max. Bits can be read and written. • Discrete Input: Discrete inputs form a bit array with a length of 65536 bits max. The bits can only be read. • Holding Register: The registers form an array of words (16 bit values) with a length of 65536 words max. The register can be read and written. • Input Register: The registers form an array of words (16 bit values) with a length of 65536 words max. The registers can only be read.

5.2 ModbusTCP server

5.2.1 ModbusTCP server applications

General information

It is useful for the configuration to execute first tests with the Modbus test clients. Test client examples available in the internet for free (date 09.2008):

- IPC@CHIP Modbus Client V1.4
- Modbus TCP/IP Request Tool Pro

The following chapters describe the configuration of a TCP connection with

- **Siemens WinCC-flexible 2007** ([Configuring with Siemens WinCC-flexible 2007 on page 198](#)) and
- **Pro-face GP-Pro EX2.2** ([Configuring with Pro-face GP-Pro EX 2.2 on page 202](#)).



The interfaces of third-party components can be subject to modifications. The use cases are thus primarily for improving knowledge. Bosch Rexroth does not guarantee the correctness of the information given.

Storage location for the example projects

The examples of the visualization systems Siemens WinCC-flexible and Pro-face GP-Pro EX2.2 are located in the AddOn directory of the IndraWorks installation DVD.

Addressing data from a PLC project

In the PLC project, data ranges for the register and bit data are transferred to function block instance IL_ModbusTCPServer using the pointer.



The data ranges of the Modbus TCP server are accessed via the following addressing by the Modbus client

Register data range:

Addressing the "Holding Register" data using **%MW** (read and write access) or the "Input Register" data using **%IW** (only read access).

Bit data range:

Addressing the "Coil" data using **%M** (read and write access) or the "Discrete Input" data using **%I** (only read access).

Bit data

In the test project, the bit data range was assigned as array of bytes from where every single bit can be addressed continuously.



No ARRAY OF BOOL can be used for the bit data range since 8 bits of memory are assigned to a variable of BOOL in the IndraLogic.

The individual bits of a byte can be copied on variables of type BOOL (and vice versa)

The address of a variable from a bit data range corresponds to the bit number from the byte array used for bit data.

Example:

Bit data (%M)

%M0 address corresponds to the **BitData[0].0** variable

%M7 address corresponds to the **BitData[0].7** variable

%M8 address corresponds to the **BitData[1].0** variable

%M12 address corresponds to the **BitData[1].4** variable

Register data

The register data range can consist of up to 65536 words and can be organized by the user without any restrictions.

For a better overview on the register data, a structure containing arrays of different PLC data types was used for the register data range in the test project. This structure was created in such a way that the individual data type areas are always provided with a length of 1000 words and do therefore begin with a defined word address.

Example of a data structure for Register_Data

```

TYPE REGISTER_DATA :
STRUCT
  arWord:   ARRAY[0..999]OF WORD;      // %MW0000 - %MW0999
  arInt:    ARRAY[0..999]OF INT;       // %MW1000 - %MW1999
  arUInt:   ARRAY[0..999]OF UINT;      // %MW2000 - %MW2999
  arDint:   ARRAY[0..499]OF DINT;      // %MW3000 - %MW3999
  arUdint:  ARRAY[0..499]OF UDINT;     // %MW4000 - %MW4999
  arReal:   ARRAY[0..499]OF REAL;      // %MW5000 - %MW5999
  arString: ARRAY[0..99]OF STRING(19); // %MW6000 - %MW6999
  // String(19): because 19 byte string + 1 byte end sign
  // together 7000 WORDs
END_STRUCT
END_TYPE

```

Text string

Texts (STRING) can also be transferred via the register data range.



In the IndraLogic, the length of a STRING (in bytes) is always the given length + 1 byte of end ID

To define the STRINGS with their total word length, the length is always to be specified with an odd number of bits

(e.g. string(19) => 19 byte string + 1 byte End ID = 20 byte = 10 words)

The address of a variable from the register data range results from the design of the structure used for the register data.

If the previously data structure is used for **RegisterData**, the following addresses result:

Example:

Register data (%MW)

%MW0 address corresponds to the **RegisterData.arWord[0]** variable

%MW1000 address corresponds to the **RegisterData.arInt[0]** variable

%MW1001 address corresponds to the **RegisterData.arInt[1]** variable

%MW5000 address corresponds to the **RegisterData.arReal[0]** variable

%MW5002 address corresponds to the **RegisterData.arReal[1]** variable

%MW6000 address corresponds to the **RegisterData.arString[0]** variable

%MW6010 address corresponds to the **RegisterData.arString[1]** variable

PLC program

Declaration

```

PROGRAM ModbusCom
// Run in a low priority task
VAR

```

RIL_ModbusTCP.library

```

fb_IL_ModbusTCPServerType02: IL_ModbusTCPServerType02; // Instance of FB IL_ModbusTCPServerType02
bEnableModbusSrv:      BOOL := TRUE;           // Enable FB IL_ModbusTCPServerType02
wPortNumber:          WORD := 502;           // Modbus server port - 502 is Modbus default
arbyBitData:          ARRAY[0..8191] OF BYTE; // Coil data area for FC2, FC5 and FC15
arbyDiscreteInputs:   ARRAY[0..8191] OF BYTE; // Discrete Input data area for FC1 (read only)
RegisterData:         REGISTER_DATA;         // Holding Register data area for FC3, FC6, FC16 and FC23
InputRegisters:       REGISTER_DATA;         // Input Register data area for FC4 (read only)
bInOperation:         BOOL;                  // FB IL_ModbusTCPServer is in operation
bError:               BOOL;                  // FB IL_ModbusTCPServer quits operation with error
ErrorID:              ERROR_CODE;            // general information about the error
ErrorIdent:           ERROR_STRUCT;          // detailed information about the error
Stats:               IL_MODBUS_STATS;        // connection statistics
END_VAR

```

Implementation part

```

fb_IL_ModbusTCPServerType02(
    Enable := bEnableModbusSrv,
    Port := wPortNumber,
    CoilData:= ADR(arbyBitData),
    SizeOfCoilData:= SIZEOF(arbyBitData),
    InputData:= ADR(arbyDiscreteInputs),
    SizeOfInputData:= SIZEOF(arbyDiscreteInputs),
    HoldingRegisterData:= ADR(RegisterData),
    SizeOfHoldingRegisterData:= SIZEOF(RegisterData),
    InputRegisterData:= ADR(InputRegisters),
    SizeOfInputRegisterData:= SIZEOF(InputRegisters),
    InOperation => bInOperation,
    Error => bError,
    ErrorID => ErrorID,
    ErrorIdent => ErrorIdent,
    Stats => Stats);

```

Configuring with Siemens WinCC-flexible 2007

Communication settings To configure the connection in WinCC-flexible, the following settings are required:

1. Open **Communication** ► **Connections** in the project tree menu of the operating device
2. Enter the name of the connection (comment is optional)
Select the communication driver **"Modicon MODBUS TCP/IP"**
3. Subsequently, the communication parameters are to be defined in the dialog

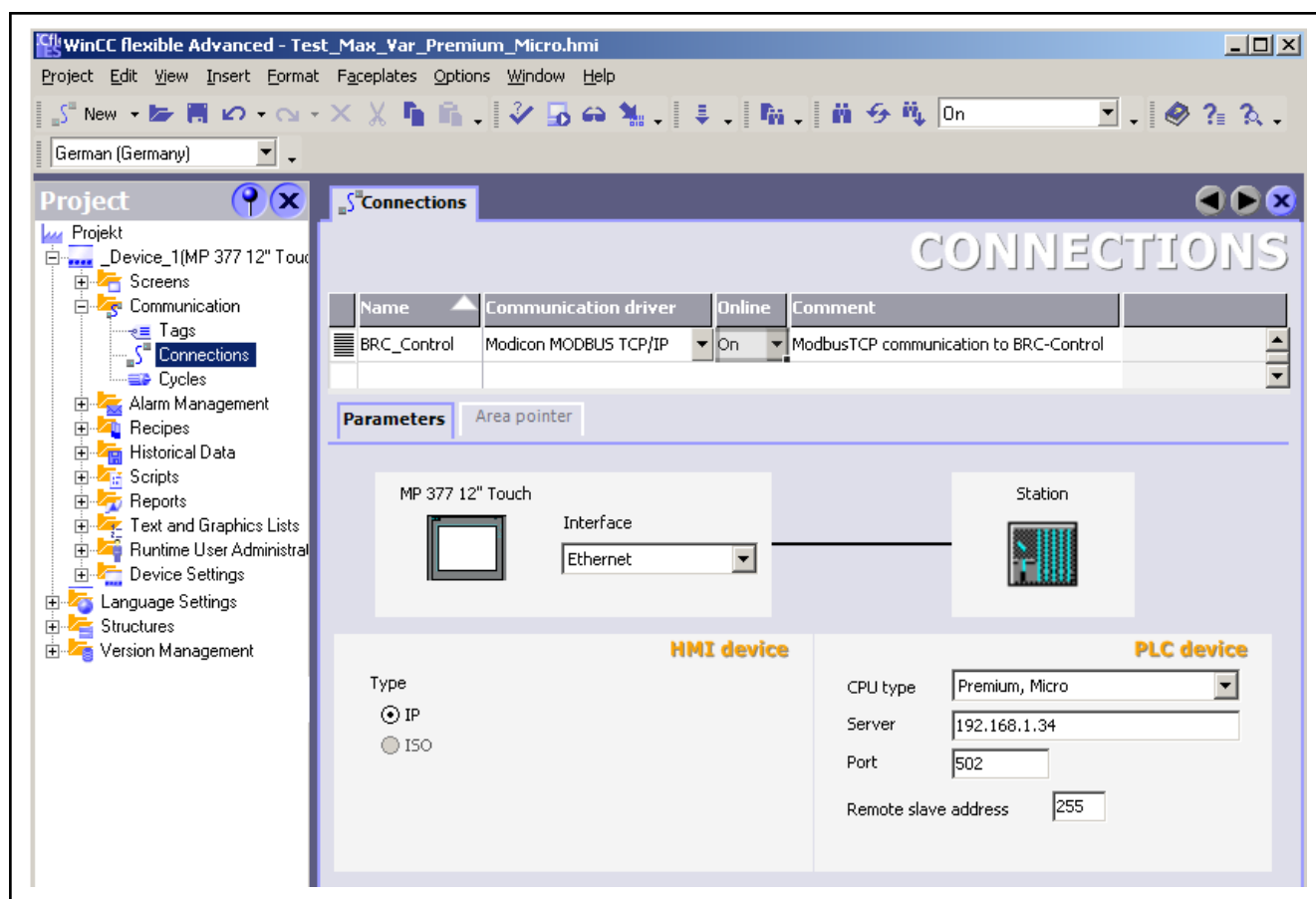


Fig. 5-2: WinCC-flexible Modbus TCP communication parameters

Section	Name	Value	Description
HMI device	Interface	Ethernet	Selection of the communication interface
	Type	IP	Only IP is supported
PLC device	CPU type	Premium, Micro	Schneider Modicon control type operating without address offset and normal bit counting (refer to the Siemens user manual "WinCC-flexible 2007 Communication Volume 2 - Chapter 6")
	Server	"192.168.1.34"	IP address of the control
	Port	"502"	Port of the communication connection (port of the IL_ModbusTCP-Server function block instance)
	Remote slave address	"255"	Maintain default setting (255)! Irrelevant for the Modbus TCP - refer to "MODBUS Messaging on TCP/IP Implementation Guide V1.0b" under www.Modbus.org

Tab. 5-1: WinCC-flexible Modbus TCP communication parameters

Variable Declaration

After configuring the connection, the variables can be declared.

Therefore, the **Communication / Tags** menu is to be opened in the project tree of the operating device. Subsequently, a dialog is displayed and the variables required can be assigned.

Example of a variable declaration in WinCC-flexible 2007:

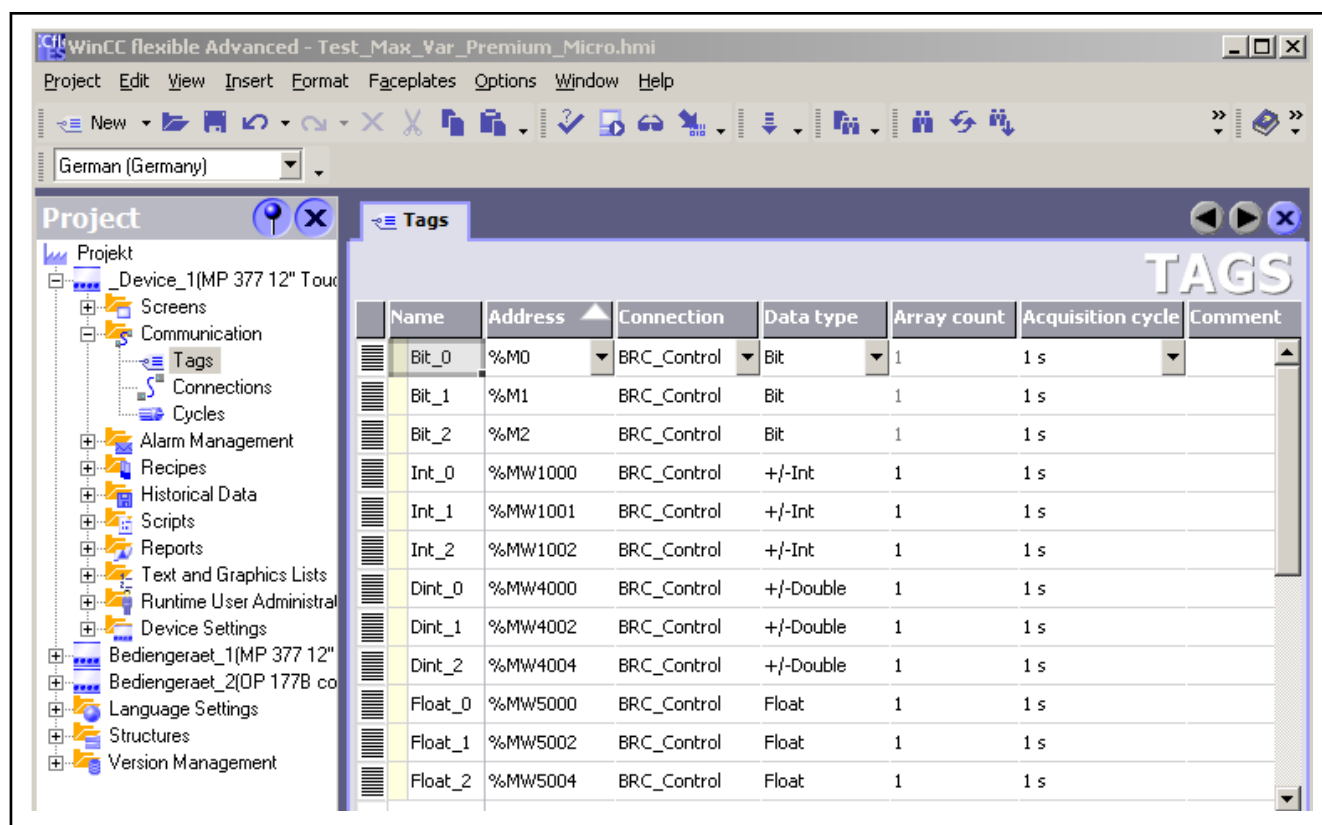


Fig. 5-3: WinCC-flexible variable declaration - Example

WinCC - flexible variable declaration

Input field	Example	Description
Name	"Bit_0"	Variable name
Address	"%M0"	Modbus address of the variable ("Addressing data from a PLC project" on page 196)
Connection	"BRC_Control"	Connection name
Data type	"Bit"	WinCC-flexible variable data type (tab. 5-3 "WinCC-flexible data types/IndraLogic data types" on page 200)
Array count	"1"	Number of array elements if variable is to be assigned as array
Acquisition cycle	"1s"	Update rate of the variable values are to be read from the control

Tab. 5-2: WinCC - flexible variable declaration

In WinCC-flexible 2007, the following data types are available. The corresponding IndraLogic data types are listed below.

WinCC - flexible data types

WinCC - flexible	IndraLogic	Description
Int	UINT	Unsigned integer length 16 bit Value range: 0 to 65535
+/-int	INT	Signed integer length 16 bit Value range: -32768 to 32767

WinCC - flexible	IndraLogic	Description
Double	UDINT	Unsigned double integer length 32 bit Value range: 0 to 4294967295
+/-double	DINT	Signed double integer length 32 bit Value range: -2147483648 to 2147483647
Float	REAL	Floating point length 32 bit Value range: 1.175494351e-38F to 3.402823466e+38F
Bit	Single bit of a byte	Single bit length 1 bit ("Bit data" on page 196)
16 bit group	WORD	WORD length 16 bits Note: For 16 bit groups, the bit counting in WinCC-flexible is vice versa to the one in IndraLogic
ASCII	STRING	The string length (in bytes) is to be configured in WinCC-flexible via the Tag Properties. ("Text string" on page 197)

Tab. 5-3: WinCC-flexible data types/IndraLogic data types

The declared variables can now be used for different WinCC-flexible functionalities.

Example:

Displaying and entering different variables

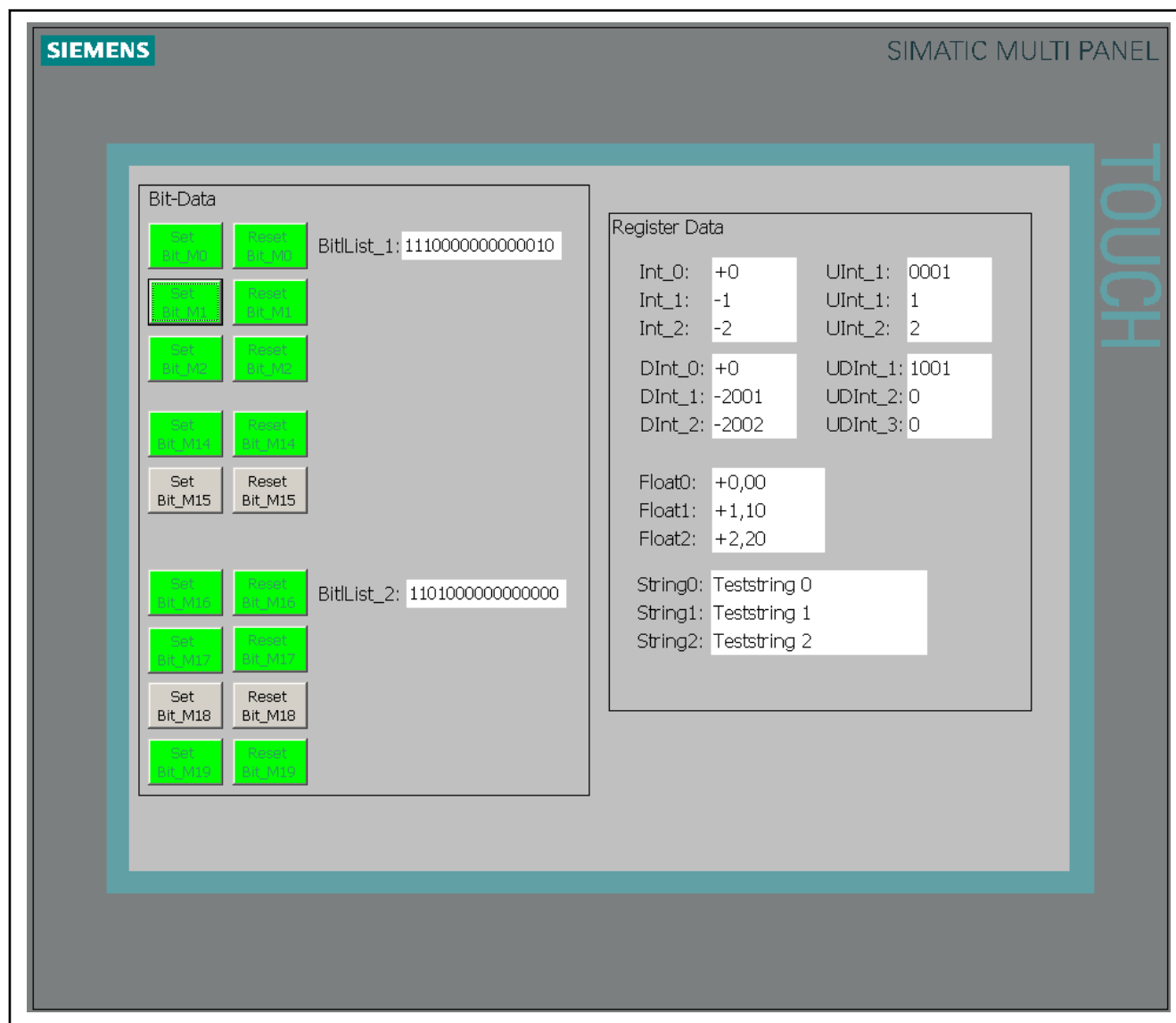


Fig. 5-4: WinCC-flexible - Visualization example

Configuring with Pro-face GP-Pro EX 2.2

Communication settings

The following settings are required to configure the connection in Pro-face GP-Pro EX 2.2:

1. Select the **Device/PLC** entry in the **"System Settings"** menu
2. Execute the **"Add Device/PLC"** link in the working area of the window
3. Subsequently, the following settings are to be carried out in the **"Add Device/PLC"** dialog:

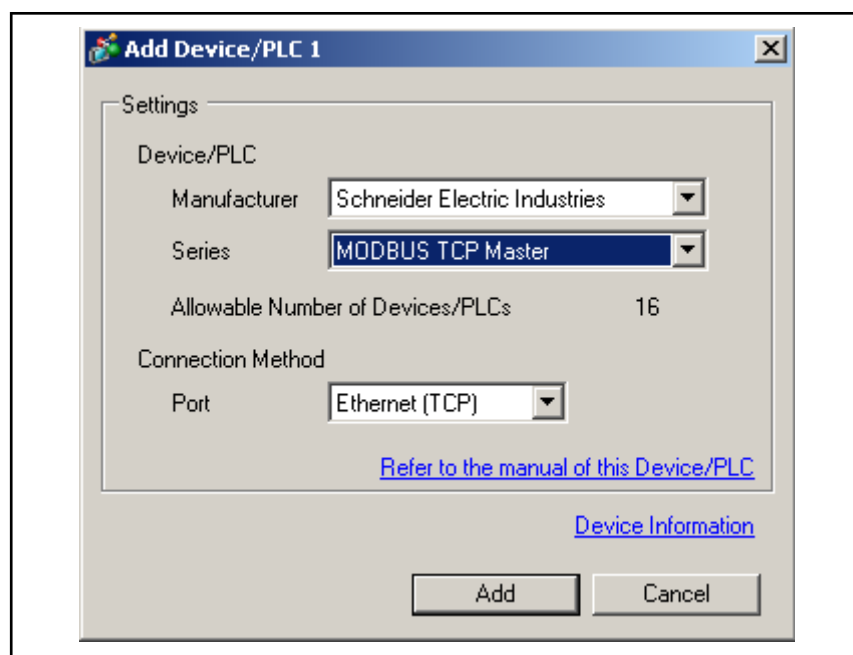


Fig. 5-5: GP-Pro EX communication settings - Device selection

Drop-down menu	Selection
Manufacturer	Schneider Electric Industries
Series	MODBUS TCP master
Port	Ethernet (TCP)

Tab. 5-4: Communication settings - Device selection

Confirm selection with <Add>

- Enter the connection name in the extended window of the working range under "Device Name" and click on  "Settings"

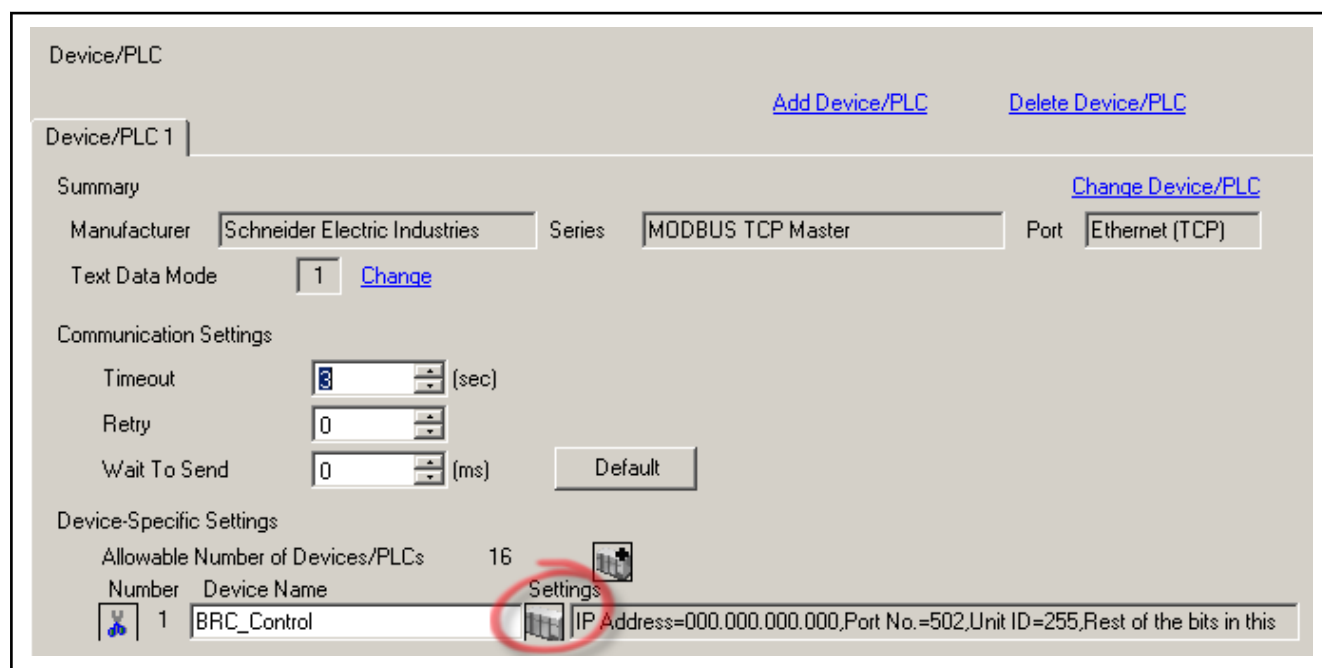


Fig. 5-6: GP-Pro EX communication settings - Working range

5. Subsequently, the following settings are to be carried out in the "Equipment Configuration" register in the "Individual Device Settings" dialog:

Individual Device Settings

BRC_Control

Equipment Configuration | Max Query

Equipment Address

IP Address: 192.168.1.34

Port No.: 502

Unit ID: 255

Bit manipulation (set/reset) to Holding Register

Rest of the bits in this word: ☐ Clear ☒ Do not clear

Note on when selecting "Do not clear":
If the ladder program writes data to Holding Register during the read/write process, the resulting data may be incorrect.

☒ IEC61131 Syntax

Address Mode: 0-based (Default)

If you change the series, please reconfirm all address settings.

Variables

Double Word word order: Low word first(L/H)

☐ Low Security Level

Default

OK (0) Cancel

Fig. 5-7: GP-Pro EX communication settings - Device parameters

Section	Name	Value	Description
Equipment Address	IP Address	"192.168.1.34"	IP address of the control
	Type	"502"	Port of the communication connection (port of the IL_ModbusTCPServer function block instance)
	Unit ID	"255"	Maintain default setting (255)! not relevant for the Modbus TCP - refer to "MODBUS Messaging on TCP/IP Implementation Guide V1.0b" on www.Modbus-IDA.org
Bit manipulation to Holding Register	Reset of the bits in this word	"Do not clear"	Maintain default setting (Do not clear)

Section	Name	Value	Description
IEC61131 Syntax	Address Mode	"0-based (default)"	Activate IEC61131 Syntax and maintain Address Mode for the default setting (0-based)
Variables	Double Word word order	"Low word first (L/H)"	Word order within a double word

Tab. 5-5: GP-Pro EX communication settings - Device parameters

The "MaxQuery" register defines the maximum number of data that can be packed in the Modbus request. The default settings remain.

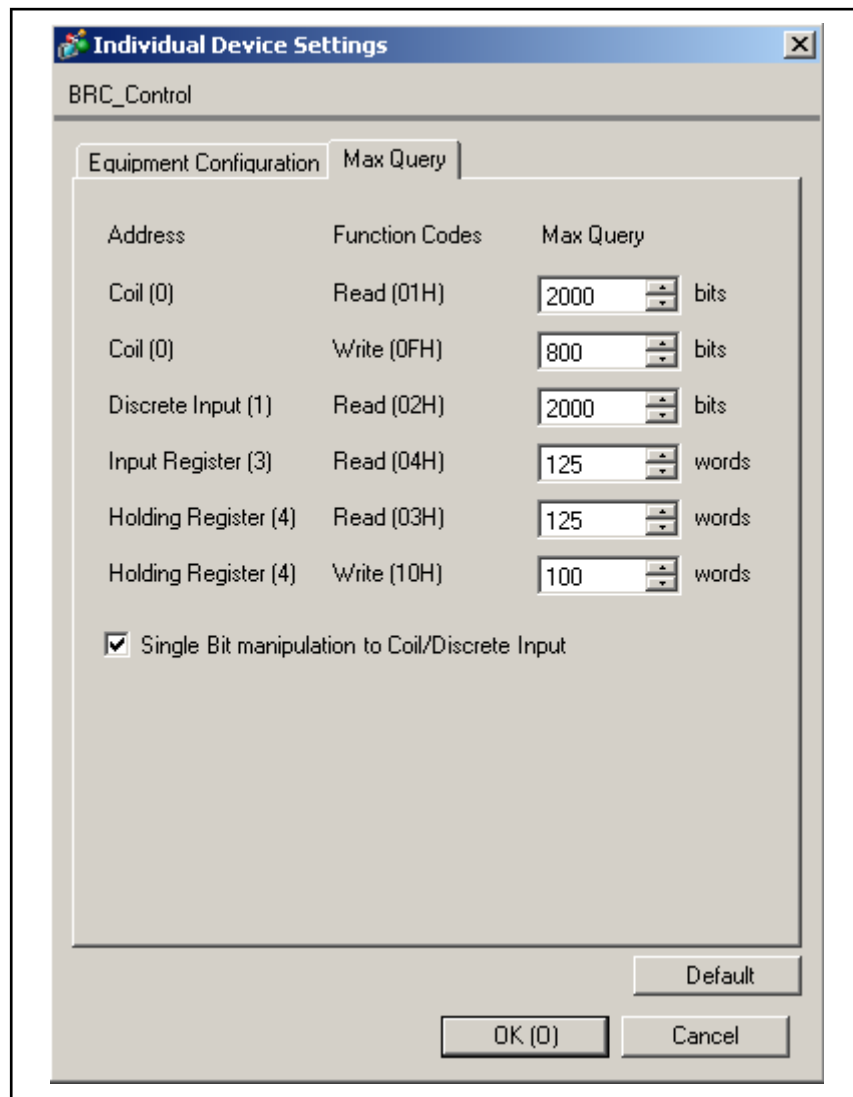
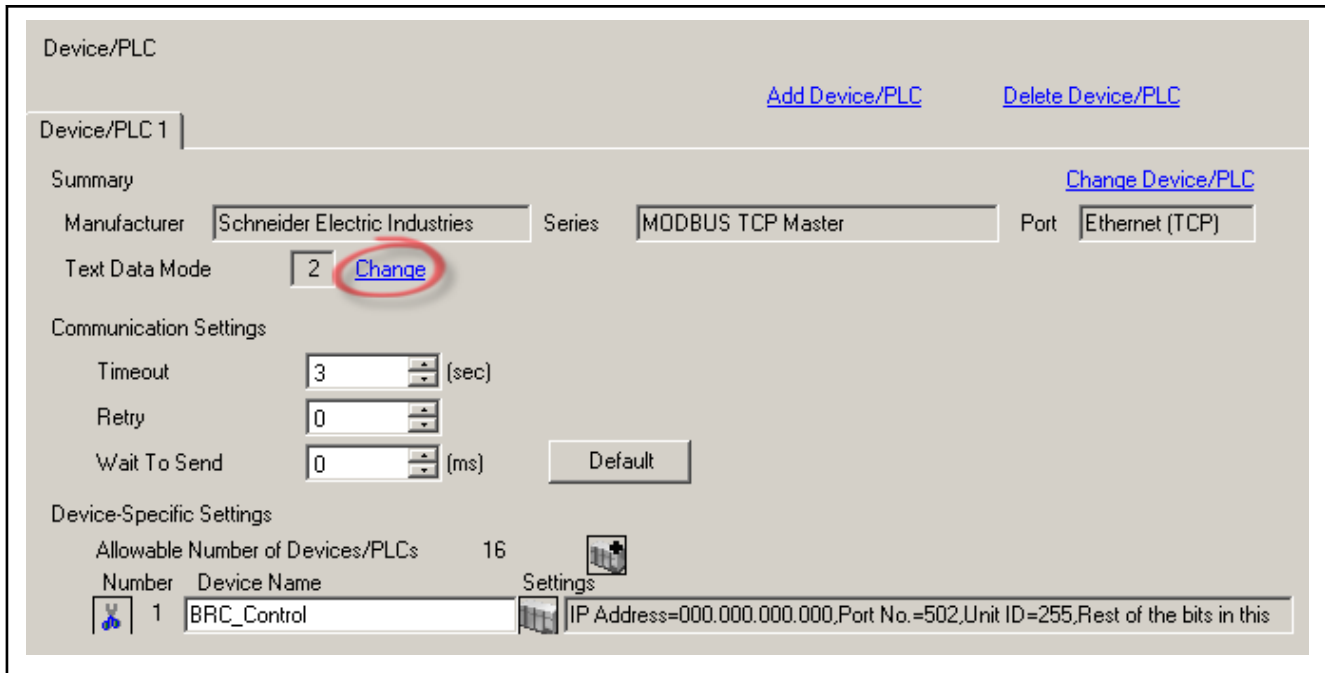


Fig. 5-8: GP-Pro EX communication settings - Device parameters

Confirm selection with <OK>.

- The "Text Data Mode" is still to be adapted to allow interpretation of texts. Therefore, click on the **"Change"** link in the working area window.



Device/PLC

[Add Device/PLC](#) [Delete Device/PLC](#)

Device/PLC 1

Summary [Change Device/PLC](#)

Manufacturer Series Port

Text Data Mode [Change](#)

Communication Settings

Timeout (sec)

Retry

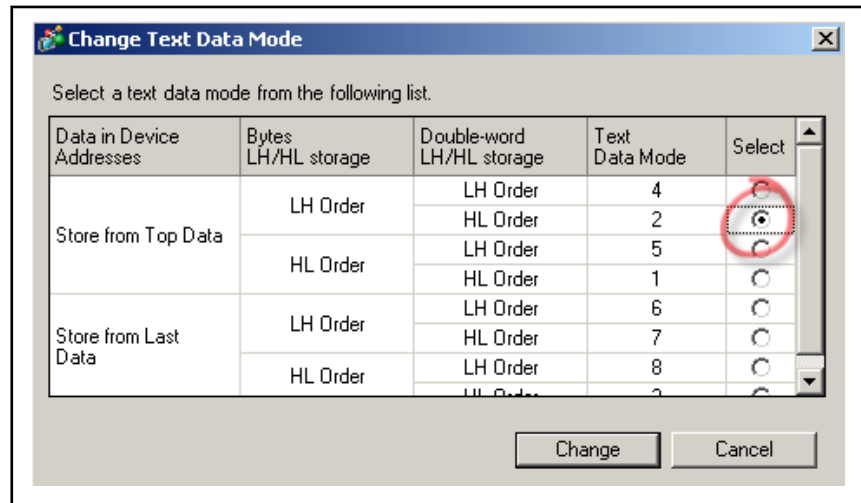
Wait To Send (ms)

Device-Specific Settings

Allowable Number of Devices/PLCs 16

Number	Device Name	Settings
1 1	BRC_Control	Settings IP Address=000.000.000.000,Port No.=502,Unit ID=255,Rest of the bits in this

Fig. 5-9: GP-Pro EX communication settings - Working range
Subsequently, select 2 in the "Change Text Data Mode" dialog and confirm with <Change>.



Change Text Data Mode

Select a text data mode from the following list.

Data in Device Addresses	Bytes LH/HL storage	Double-word LH/HL storage	Text Data Mode	Select
Store from Top Data	LH Order	LH Order	4	☐
		HL Order	2	☒
	HL Order	LH Order	5	☐
		HL Order	1	☐
Store from Last Data	LH Order	LH Order	6	☐
		HL Order	7	☐
	HL Order	LH Order	8	☐
		HL Order	9	☐

Fig. 5-10: GP-Pro EX communication settings - Text data mode

Using addresses

After configuring the connection, the addresses can be used directly in different GP-Pro EX functionalities, refer to ["Addressing data from a PLC project"](#) on page 196.

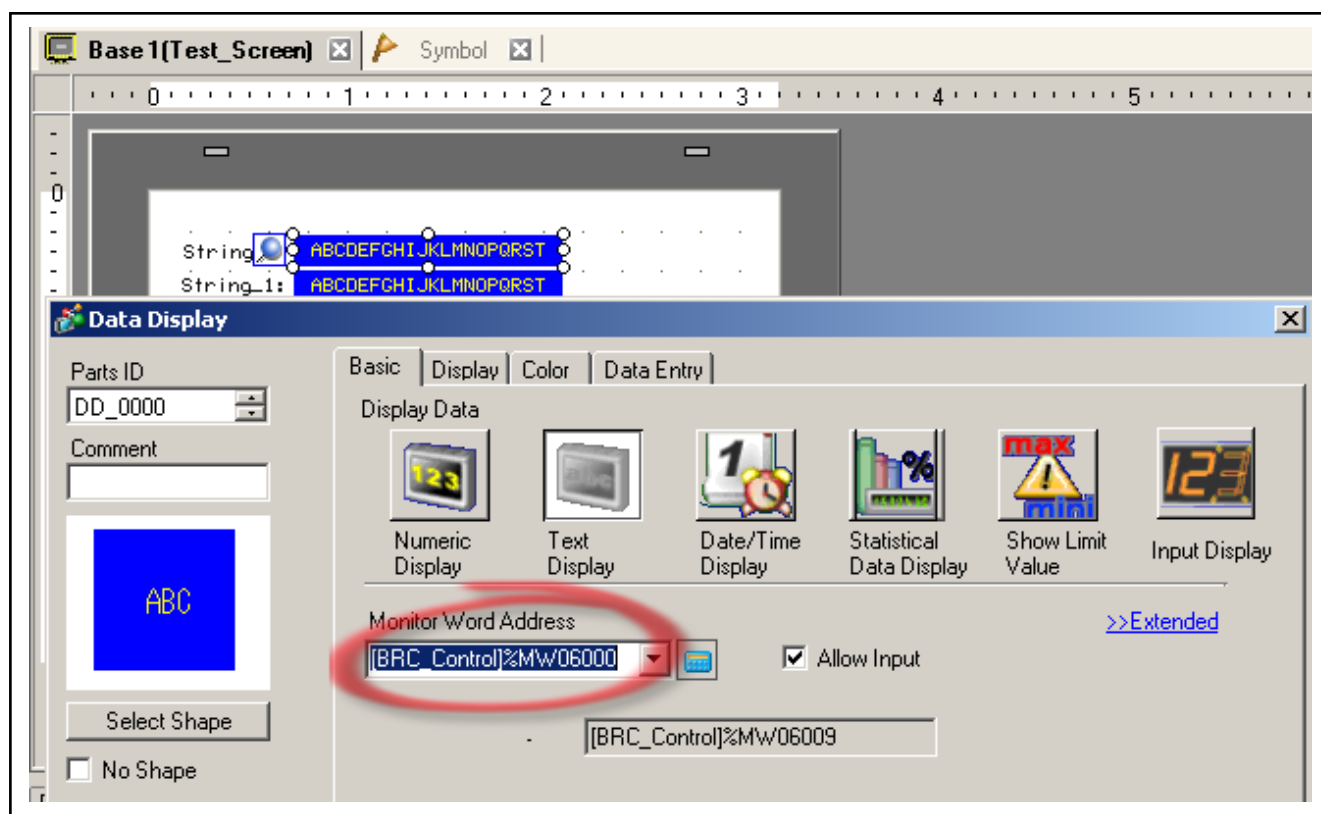


Fig. 5-11: GP-Pro EX - Direct address use

Variable Declaration

Symbol variables of type bit address and word address can also be declared. For details, please refer to ["Addressing data from a PLC project" on page 196](#). The address content can be used in different GP-Pro EX functionalities via these symbol variables.

To declare the symbol variables in the working area of the window, open **Common Settings ▶ Symbol Variables** in the menu.

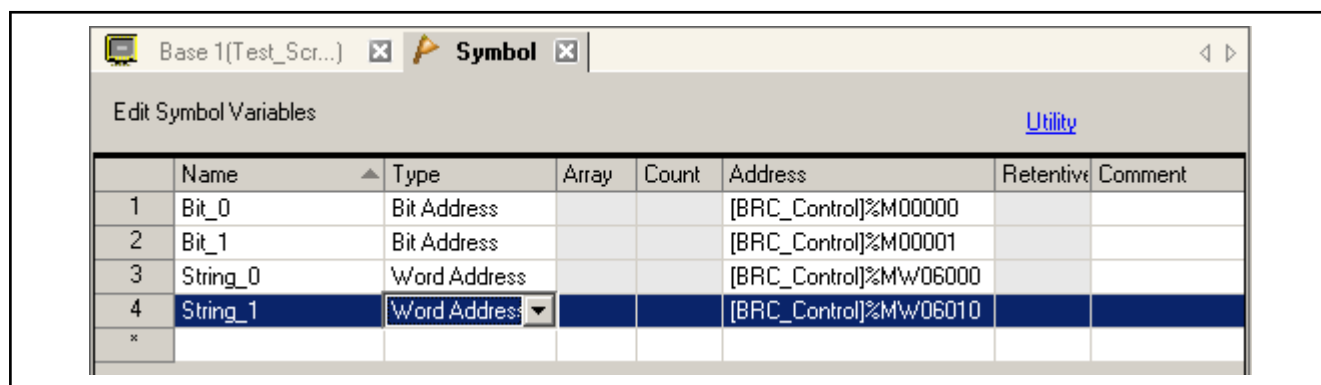


Fig. 5-12: GP-Pro EX - Create a symbol variable

After creating the symbol variable, it can be used as "placeholder" for the address.

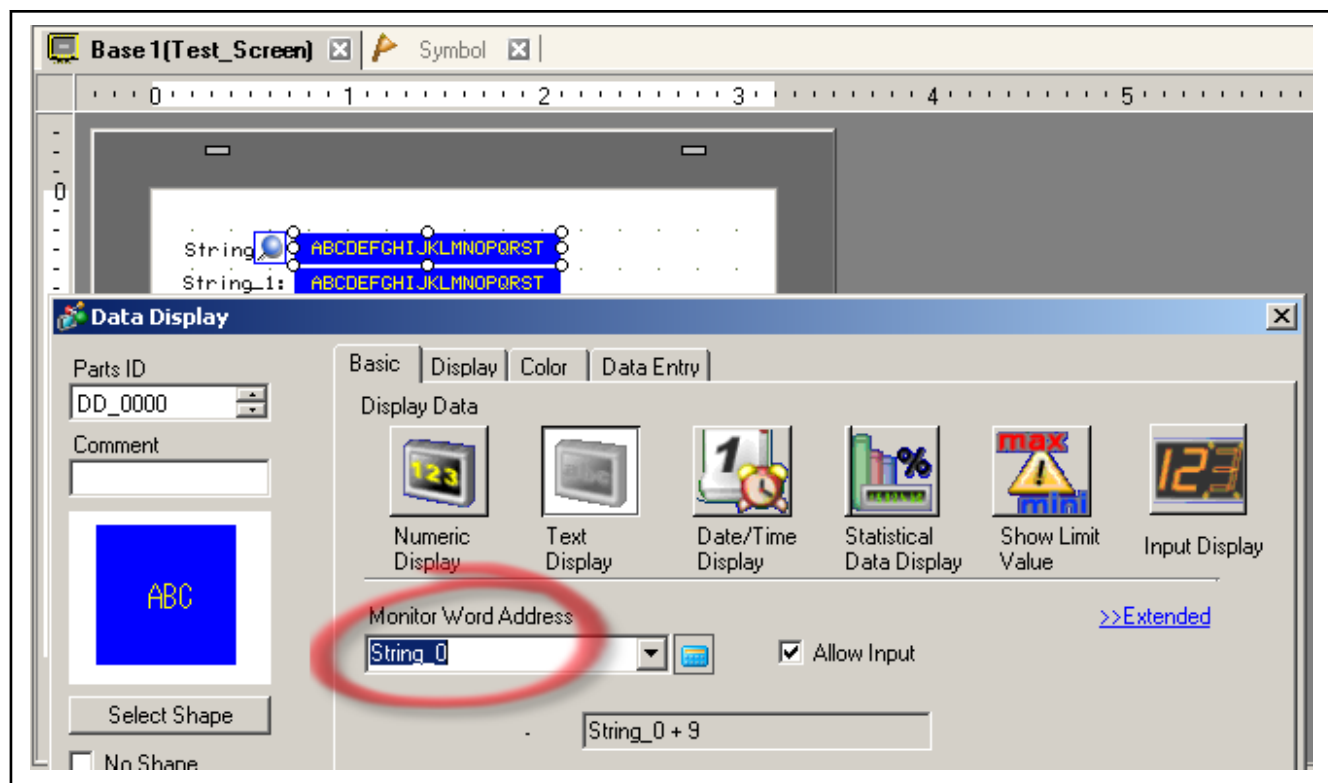


Fig. 5-13: GP-Pro EX - Using symbol variables



The number of the registers to be interpreted is defined via the data type used in the individual display fields.

For display fields of type "Text Display", the number of characters to be displayed are defined via the number of the registers to be interpreted.

Example:

Displaying and entering different variables

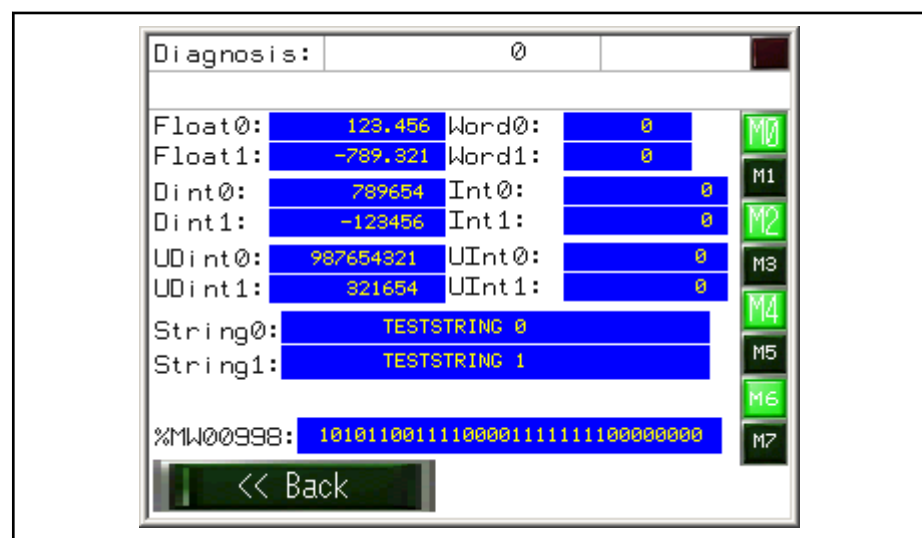


Fig. 5-14:

5.2.2 IL_ModbusTCPServer

Brief description A Modbus TCP slave is implemented in the IL_ModbusTCPServer function block. Discrete Inputs, Coils, Input Registers and Holding Registers are supported with read and write access. Arrays are used for the data exchange.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/server

Tab. 5-6: Library assignment

Interface description

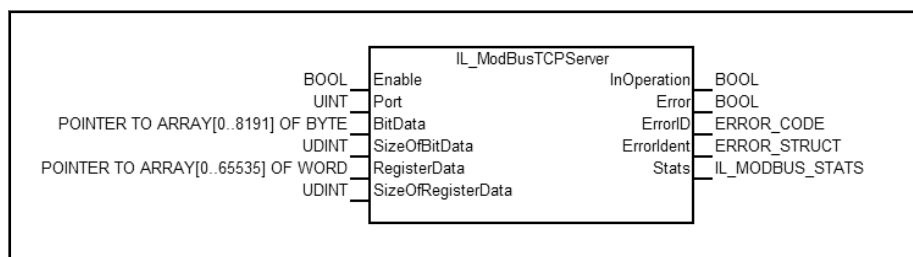


Fig. 5-15: IL_ModbusTCPServer function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Port	UINT	TCP port at which incoming connections are expected.
	BitData	POINTER TO ARRAY[0..8191] OF BYTE	Pointer to bit data.
	SizeOfBitData	UDINT	Number of bytes in BitData (bit number/8).
	RegisterData	POINTER TO ARRAY[0...65535] OF WORD	Pointer to register data in word width.
	SizeOfRegisterData	UDINT	Number of bytes in the register data (register number * 2).
VAR OUTPUT	InOperation	BOOL	The Modbus server runs and accepts incoming connections.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 212 .
	Stats		It contains counter for data packages and errors. IL_MODBUS_STATS, page 218

Tab. 5-7: IL_ModbusTCPServer interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Port	UINT	0	65535	502	Rising edge at "Enable"

Name	Type	Min. value	Max. value	Default value	Effective
BitData	POINTER	0	N. def.	0	Rising edge at "Enable"
SizeOfBitData	UDINT	0	8191	0	Rising edge at "Enable"
RegisterData	POINTER	0	N. def.	0	Rising edge at "Enable"
SizeOfRegisterData	UDINT	0	131070	0	Rising edge at "Enable"

Tab. 5-8: Input behavior of the IL_ModbusTCPServer

Functional Description

The IL_ModbusTCPServer function block implements a Modbus slave that can connect up to 16 Modbus masters. The function block is passive during the data exchange. Being passive means that the Modbus master specifies which data is read and which data is written. Discrete Inputs, Coils, Input Registers and Holding Registers are supported.



In contrast to the Modbus specification, the respective Discrete Inputs, Coils as well as Input Registers and Holding Registers are imaged on the same data range.

In case more than 16 devices try to connect simultaneously, the connection to the most previous data exchange is aborted before a new connection is accepted. This behavior complies with the Modbus TCP specification. This should be avoided in practice. Therefore, it is useful to open a second Modbus TCP server on a different port.

Enable, InOperation

"Enable" is the processing enabled of the function block. The function block is active as long as "Enable" is TRUE. At a rising edge at "Enable", all inputs are applied and the function block is initialized. After the initialization is completed, the output is set to "InOperation" to indicate that the incoming Modbus TCP connections are accepted and can now be processed. At a falling edge at "Enable", the existing connections are closed and the port is shared.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are only set if errors occur while processing the function block. Errors caused by the Modbus TCP protocol do not generate any output at the error outputs. The "NumExceptions" field in the "Stats" output is used for that purpose.

Port

Generally, the Modbus TCP uses the port 502, but a Modbus TCP server can be optionally opened on a different port as well. Therefore, the port number to be used has to be written on the "Port" input before the function block is enabled.

BitData, SizeOfBitData

The "BitData" input corresponds to the Discrete Inputs and Coils defined in the Modbus. Coils form a bit array with a length of 65536 bits max. Bits can be read and written. The data exchange is executed via the array assigned at the "BitData" input. Each array byte represents 8 bits.

The array size in byte is to be assigned at the "SizeOfBitData" input. The SI-ZEOF() operator can be used for this. The function block checks whether the incoming requests are within the assigned array using the "SizeOfBitData".



It is useful to define an own data structure to access the bits. The pragma `{bitaccess}` in the IndraLogic 1.x or `{bit}` allows an access on the individual bits in IndraLogic 2G. For details, refer to the application description of the project conversion [DOK-MLC***-PROCONV*V15-APRS](#), chapter **Bit access**.

RegisterData, SizeOfRegisterData

The "RegisterData" input corresponds to the Input Registers and Holding Registers defined in the Modbus. The registers form an array of words (16 bit values) with a length of 65536 words max. Data is exchanged via the array applied at the "RegisterData" input.

The array size in bytes has to be applied at the "SizeOfRegisterData" input. The `SIZEOF()` operator can be used for this. The function block checks whether the incoming requests are within the assigned array using the "SizeOfRegisterData".



It is useful to define an own data structure to access the registers. The alignment of the data (16 bit) has to fit into words. Refer to [chapter 5.2.1 "ModbusTCP server applications" on page 196](#).

Stats

The "Stats" output contains a data structure indicating the status and statistic information on the running communication and the IP addresses of the connected clients. Read and write access is counted in the corresponding fields. Protocol errors are only counted in the "NumExceptions" field. If the maximum number of the counter is exceeded, it restarts with 0. The maximum value of a counter is $2^{32}-1$.

Supported Modbus function codes

Function codes are used by the Modbus master to access the Modbus slave. Knowledge about the codes supported helps to find the errors. It is not required for the normal operating mode though. The following function codes are supported:

Function code	Name	Description
1	Read Coils	Reads a number of bits from "BitData" starting from an initial address
2	Read Discrete Inputs	Reads a number of bits from "BitData" starting from an initial address
3	Read Holding Registers	Reads a number of registers from "RegisterData" starting from an initial address
4	Read Input Registers	Reads a number of registers from "RegisterData" starting from an initial address
5	Write Single Coil	Writes a bit into "BitData" starting from an initial address
6	Write Single Register	Writes a register into "RegisterData" starting from an initial address
15	Write Multiple Coils	Writes a number of bits into "BitData" starting from an initial address
16	Write Multiple Registers	Writes a number of registers into "RegisterData" starting from an initial address

Function code	Name	Description
23	Read/Write Multiple Registers	Executes a combined read/write operation on "RegisterData"
43/14	Read Device Identification	Reads the default device information of the Modbus TCP server: • VendorName: "Bosch Rexroth" • ProductCode: "IL_ModbusTCPServer" • MajorMinorRevision: "01Vxx"

Tab. 5-9: Supported Modbus function codes



The function codes 1, 2, 5, 15 access a storage area assigned at the "BitData" input.



The function codes 3, 4, 6, 16 access a storage area assigned at the "RegisterData" input.

Troubleshooting

The function block uses the F_RELATED_TABLE, 16#170x error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#1800	16#0000	The "BitData" input is a zero pointer.
INPUT_INVALID_ERROR (16#0001)	16#1801	16#0000	The "RegisterData" input is a zero pointer.
INPUT_INVALID_ERROR (16#0001)	16#1802	16#0000	The "SizeOfBitData" input is 0.
INPUT_INVALID_ERROR (16#0001)	16#1803	16#0000	The "SizeOfRegisterData" input is 0.
INPUT_INVALID_ERROR (16#0001)	16#1804	16#0000	The "SizeOfRegisterData" input is odd.
RESOURCE_ERROR (16#0003)	16#1805	16#0000	Could not open socket.
RESOURCE_ERROR (16#0003)	16#1806	16#0000	Could not accept incoming connection.
RESOURCE_ERROR (16#0003)	16#1807	16#0000	The port is assigned.
RESOURCE_ERROR (16#0003)	16#1808	16#0000	Port listen failed.

Tab. 5-10: Error codes of the IL_ModbusTCPServer

5.2.3 IL_ModbusTCPServerType02

Brief description

A Modbus TCP slave is implemented in the IL_ModbusTCPServerType02 function block. Discrete Inputs, Coils, Input Registers and Holding Registers are supported with read and write access. The data exchange is executed via 4 separate arrays for the individual data ranges.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/server

Tab. 5-11: Library assignment

Interface description

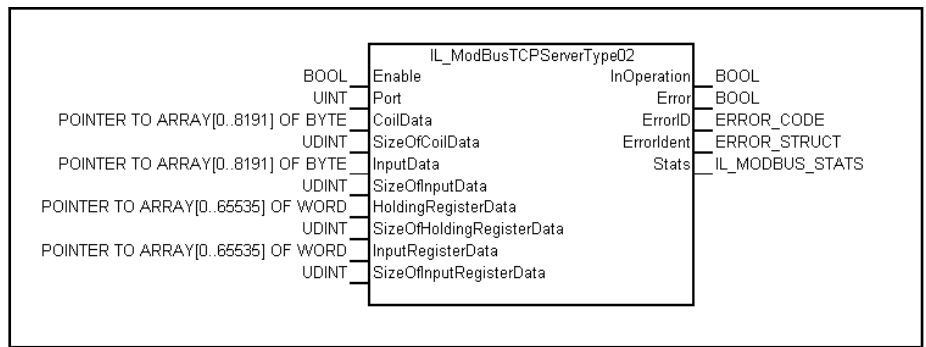


Fig. 5-16: IL_ModbusTCPServerType02 function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Port	UINT	TCP port at which incoming connections are expected.
	CoilData	POINTER TO ARRAY[0..8191] OF BYTE	Pointer to coil data (bit data range is neither read-protected nor write-protected).
	SizeOfCoilData	UDINT	Number of bytes in CoilData (bit number/8).
	InputData	POINTER TO ARRAY[0..8191] OF BYTE	Pointer to discrete input data (bit data range is not read-protected).
	SizeOfInputData	UDINT	Number of bytes in InputData (bit number/8).
	HoldingRegisterData	POINTER TO ARRAY[0..65535] OF WORD	Pointer to Holding register data in word width (register data range is neither read-protected nor write-protected).
	SizeOfHoldingRegisterData	UDINT	Number of bytes in the holding register data (register number * 2).
	InputRegisterData	POINTER TO ARRAY[0..65535] OF WORD	Pointer to Input register data in word width (register data range is not read-protected).
	SizeOfInputRegisterData	UDINT	Number of bytes in the Input register data (register number * 2).
VAR OUTPUT	InOperation	BOOL	The Modbus server runs and accepts incoming connections.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 217 .
	Stats		It contains counter for data packages and errors. IL_MODBUS_STATS, page 218

Tab. 5-12: IL_ModbusTCPServerType02 interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Port	UINT	0	65535	502	Rising edge at "Enable"
CoilData	POINTER	0	n. def.	0	Rising edge at "Enable"
SizeOfCoilData	UDINT	0	8191	0	Rising edge at "Enable"
InputData	POINTER	0	n. def.	0	Rising edge at "Enable"
SizeOfInputData	UDINT	0	8191	0	Rising edge at "Enable"
HoldingRegisterData	POINTER	0	n. def.	0	Rising edge at "Enable"
SizeOfHoldingRegisterData	UDINT	0	131070	0	Rising edge at "Enable"
InputRegisterData	POINTER	0	n. def.	0	Rising edge at "Enable"
SizeOfInputRegisterData	UDINT	0	131070	0	Rising edge at "Enable"

Tab. 5-13: Input behavior of the IL_ModbusTCPServerType02

Name	Type	Default value	Description
NumReadBits	UDINT	0	Number of read access on bits
NumReadRegisters	UDINT	0	Number of read access on register
NumWriteBit	UDINT	0	Number of single write access on bits
NumWriteRegister	UDINT	0	Number of single write access on register
NumWriteBits	UDINT	0	Number of multiple write access on bits
NumWriteRegisters	UDINT	0	Number of multiple write access on register
NumReadWriteRegs	UDINT	0	Number of combined read/write access on register
NumExceptions	UDINT	0	Number of Modbus errors
ConnectedClients	ARRAY[0..15] OF String(16)		IP addresses of the clients connected

Tab. 5-14: IL_MODBUS_STATS data structure

Functional Description

The IL_ModbusTCPServerType02 function block implements a Modbus slave that can connect up to 16 Modbus masters. The function block is passive during the data exchange. Being passive means that the Modbus master (client) specifies which data is read and which data is written. Discrete Inputs, Coils, Input Registers and Holding Registers are supported. As in the Modbus specification, the Discrete Inputs, Coils as well as Input Registers and Holding Registers are imaged on separate data ranges.

In case more than 16 devices try to connect simultaneously, the connection to the most previous data exchange is aborted before a new connection is

	accepted. This behavior complies with the Modbus TCP specification. This should be avoided in practice. Therefore, it is useful to open a second Modbus TCP server on a different port.
Enable, InOperation	"Enable" is the processing enabled of the function block. The function block is active as long as "Enable" is TRUE. At a rising edge at "Enable", all inputs are applied and the function block is initialized. After the initialization is completed, the output is set to "InOperation" to indicate that the incoming Modbus TCP connections are accepted and can now be processed. At a falling edge at "Enable", the existing connections are closed and the port is shared.
Error, ErrorID, ErrorIdent	If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset. The error outputs are only set if errors occur while processing the function block. Errors caused by the Modbus TCP protocol do not generate any output at the error outputs. The "NumExceptions" field in the "Stats" output is used for that purpose.
Port	Generally, the Modbus TCP uses the port 502, but a Modbus TCP server can be optionally opened on a different port as well. Therefore, the port number to be used has to be written on the "Port" input before the function block is enabled.
CoilData, SizeOfCoilData	The "CoilData" input corresponds to the coils defined in the Modbus. Coils form a bit array with a length of 65536 bits max. Bits can be read and written. The data exchange is executed via the array assigned at the "CoilData" input. Each array byte represents 8 bits. The array size in byte is to be assigned at the "SizeOfCoilData" input. The SIZEOF() operator can be used for this. The function block checks whether the incoming requests are within the assigned array using the "SizeOfCoilData".
	 It is useful to define an own data structure to access the bits. The pragma {bitaccess } allows an access on the individual bits.
InputData, SizeOfInputData	The "InputData" input corresponds to the discrete inputs defined in the Modbus. Discrete inputs form a bit array with a length of 65536 bits max. The bits can only be read. The data exchange is executed via the array assigned at the "InputData" input. Each array byte represents 8 bits. The array size in byte is to be assigned at the "SizeOfInputData" input. The SIZEOF() operator can be used for this. The function block checks whether the incoming requests are within the assigned array using the "SizeOfInputData".
	 It is useful to define an own data structure to access the bits. The pragma {bitaccess } allows an access on the individual bits.
HoldingRegisterData, SizeOfHoldingRegisterData	The "HoldingRegisterData" input corresponds to the Holding register defined in the Modbus. The registers form an array of words (16 bit values) with a length of 65536 words max. The register can be read and written. The data exchange is executed via the array assigned at the "HoldingRegisterData" input. The array size in byte is to be assigned at the "SizeOfHoldingRegisterData" input. The SIZEOF() operator can be used for this. The function block checks whether the incoming requests are within the assigned array using the "SizeOfHoldingRegisterData".



It is useful to define an own data structure to access the registers. The alignment of the data (16 bit) has to fit into words. Refer to ["Example of a data structure for Register_Data" on page 197.](#)

InputRegisterData, SizeOfInputRegisterData

The "InputRegisterData" input corresponds to the Input register defined in the Modbus. The registers form an array of words (16 bit values) with a length of 65536 words max. The registers can only be read. The data exchange is executed via the array assigned at the "InputRegisterData" input.

The array size in byte is to be assigned at the "SizeOfInputRegisterData" input. The SIZEOF() operator can be used for this. The function block checks whether the incoming requests are within the assigned array using the "SizeOfInputRegisterData".



It is useful to define an own data structure to access the registers. The alignment of the data (16 bit) has to fit into words. Refer to ["Example of a data structure for Register_Data" on page 197.](#)

Stats

The "Stats" output contains a data structure indicating the status and statistic information on the running communication and the IP addresses of the connected clients. Read and write access is counted in the corresponding fields. Protocol errors are only counted in the "NumExceptions" field. If the maximum number of the counter is exceeded, it restarts with 0. The maximum value of a counter is $2^{32}-1$.

Supported Modbus function codes

Function codes are used by the Modbus master (client) to access the Modbus slave (server). Knowledge about the codes supported helps to find the errors. It is not required for the normal operating mode though. The following function codes are supported:

Function code	Name	Description
1	Read Coils	Reads a number of bits from "CoilData" starting from an initial address
2	Read Discrete Inputs	Reads a number of bits from "InputData" starting from an initial address
3	Read Holding Registers	Reads a number of registers from "HoldingRegisterData" starting from an initial address
4	Read Input Registers	Reads a number of registers from "InputRegisterData" starting from an initial address
5	Write Single Coil	Writes a bit into "CoilData" starting from an initial address
6	Write Single Register	Writes a register into "HoldingRegisterData" starting from an initial address
15	Write Multiple Coils	Writes a number of bits into "CoilData" starting from an initial address
16	Write Multiple Registers	Writes a number of registers into "HoldingRegisterData" starting from an initial address

Function code	Name	Description
23	Read/Write Multiple Registers	Executes a combined read/write operation on "HoldingRegisterData"
43/14	Read Device Identification	Retrieves standard device information of the Modbus TCP server: - VendorName: "Bosch Rexroth" - ProductCode: "IL_ModbusTCPServerType02" - MajorMinorRevision: "01Vxx"

Tab. 5-15: Supported Modbus function codes



The function codes 1, 5, 15 access a storage area assigned at the "CoilData" input.



The function code 2 accesses a storage area assigned at the "InputData" input.



The function codes 3, 6, 16 access a storage area assigned at the "HoldingRegisterData" input.



The function code 4 accesses a storage area assigned at the "InputRegisterData" input.

Troubleshooting

The function block uses the F_RELATED_TABLE, 16#170x error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	16#00001800	16#00000000	The "CoilData" input is a zero pointer.
		16#00000001	The "InputData" input is a zero pointer.
INPUT_INVALID_ERROR (16#0001)	16#00001801	16#00000000	The "HoldingRegisterData" input is a zero pointer.
		16#00000001	The "InputRegisterData" input is a zero pointer.
INPUT_INVALID_ERROR (16#0001)	16#00001802	16#00000000	The "SizeOfCoilData" input is 0.
		16#00000001	The "SizeOfInputData" input is 0.
INPUT_INVALID_ERROR (16#0001)	16#00001803	16#00000000	The "SizeOfHoldingRegisterData" input is 0.
		16#00000001	The "SizeOfInputRegisterData" input is 0.
INPUT_INVALID_ERROR (16#0001)	16#00001804	16#00000000	The "SizeOfHoldingRegisterData" input is odd.
		16#00000001	The "SizeOfInputRegisterData" input is odd.
RESOURCE_ERROR (16#0003)	16#00001805	16#00000000	Could not open socket.
RESOURCE_ERROR (16#0003)	16#00001806	16#00000000	Could not accept incoming connection.
RESOURCE_ERROR (16#0003)	16#00001807	16#00000000	The port is assigned.
RESOURCE_ERROR (16#0003)	16#00001808	16#00000000	Port listen failed.

Tab. 5-16: Error codes of the IL_ModbusTCPTType02

5.2.4 IL_MODBUS_STATS

The IL_MODBUS_STATS structure contains status and statistic information on the running communications and the IP addresses of the connected client. Read and write access is counted in the corresponding fields. Protocol errors are only counted in the "NumExceptions" field. If the maximum number of the counter is exceeded, it restarts with 0. The maximum value of a counter is $2^{32}-1$.

Name	Type	Default value	Description
NumReadBits	UDINT	0	Number of read access on bits
NumReadRegisters	UDINT	0	Number of read access on register
NumWriteBit	UDINT	0	Number of single write access on bits
NumWriteRegister	UDINT	0	Number of single write access on register
NumWriteBits	UDINT	0	Number of multiple write access on bits
NumWriteRegisters	UDINT	0	Number of multiple write access on register
NumReadWriteRegs	UDINT	0	Number of combined read/write access on register
NumExceptions	UDINT	0	Number of Modbus errors
ConnectedClients	ARRAY[0..15] OF String(16)		IP addresses of the clients connected

Tab. 5-17: IL_MODBUS_STATS data structure

5.3 ModbusTCP client

5.3.1 Introduction and overview

The function blocks to the ModbusTCP client exchange data with a ModbusTCP server. The communication partner can either be an I/O device or another control.

The following example shows the cyclic data exchange with another control.

Two data arrays (arrays of word) have to be defined.

Additionally, one instance of each the function blocks IL_ModbusTCPClient and IL_ModbusTCPReadWriteMultipleRegisters are required.

Program:

```
// IP address of ModbusTCP server
strServerIP: STRING := '192.168.42.10';

// Data arrays
arOutData: ARRAY[100..199] OF WORD;
arInData: ARRAY[500..599] OF WORD;

// Function blocks
fbModBusTCPClient: IL_ModBusTCPClient;
fbModBusTCPReadWriteRegisters: IL_ModBusTCPReadWriteMultipleRegisters;
```

The IL_ModbusTCPClient function block establishes the connection to the ModbusTCP server.

It does not yet exchange data. Data is exchanged by other client function blocks connected to an instance of the IL_ModbusTCPClient function block via the "ModbusLink".

Program:

```

fbModBusTCPClient(Enable:=TRUE, Address:=strServerIP);

fbModBusTCPReadWriteRegisters(
  Enable := fbModBusTCPClient.InOperation,
  FirstReadRegister := 100,
  ReadQuantity := 100,
  ReadDataAddr := ADR(arInData),
  SizeOfReadData := SIZEOF(arInData),
  FirstWriteRegister := 500,
  WriteQuantity := 100,
  WriteDataAddr := ADR(arOutData),
  SizeOfWriteData := SIZEOF(arOutData)
);

```

This simplified example program establishes a ModbusTCP connection to a ModbusTCP server specified in strServerIP.

The holding registers 100 to 199 are subsequently read cyclically and the holding registers 500 to 599 are written. This example does not contain any troubleshooting.

5.3.2 IL_ModbusTCPClient

Brief description A Modbus TCP master is implemented in the IL_ModbusTCPClient function block.

It is used to establish a connection to the ModbusTCP server.

The actual ModbusTCP queries are implemented by other function blocks.

Target system	Library/folder
MLC 15VRS	RIL_ModbusTcpc/client
Basic function package PLC	

Tab. 5-18: Library assignment

Interface description

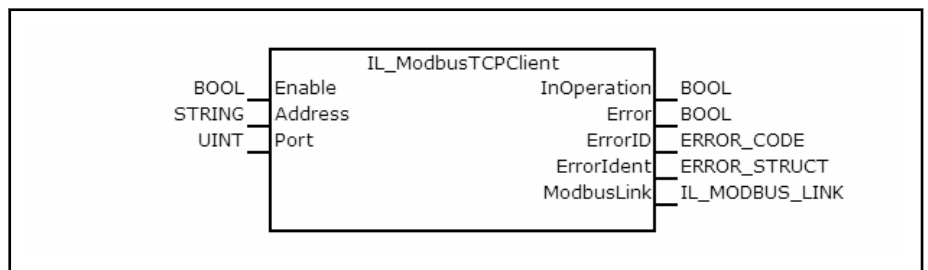


Fig. 5-17: IL_ModbusTCPClient function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	Address	STRING	IP address to be connected to.
	Port	UINT	TCP port of the ModbusTCP server.

	Name	Type	Description
VAR OUTPUT	InOperation	BOOL	The Modbus server runs and accepts incoming connections.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information See also "Troubleshooting" on page 221 .
	ModbusLink	IL_MODBUS_LINK	Handle on the existing Modbus connection is required by the client function blocks

Tab. 5-19: IL_ModbusTCPClient interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Enable"
Port	UINT	0	65535	502	Rising edge at "Enable"
UnitID	USINT	0	255	255	Rising edge at "Enable"

Tab. 5-20: Input behavior IL_ModbusTCPClient

Functional description

A Modbus TCP master is implemented in the IL_ModbusTCPClient function block.

It is used to establish a connection to the ModbusTCP server.

The actual ModbusTCP queries are implemented by other function blocks.

Enable, InOperation

At a rising edge at "Enable", the function block establishes a connection to the ModbusTCP server specified in "Address" and "Port". When the connection is established successfully, the "InOperation" output is set. The "ModbusLink" output can then be used as input for the ModbusTCP client function blocks. At a falling edge at "Enable", the connection to the ModbusTCP server is closed.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are only set if errors occur while processing the function block. Errors caused by the Modbus TCP protocol do not generate any output at the error outputs. These are reported by the ModbusTCP client function blocks.

Port

Modbus TCP normally uses port 502. If the ModbusTCP server runs on another TCP port, the port number to be used can be written to the "Port" input.

Supported Modbus function codes

Function codes are used by the Modbus master to access the Modbus slave.

Knowledge about the codes supported helps to find the errors. It is not required for the normal operating mode though.

The following function codes are supported by the Modbus TCP client function blocks:

Function code	Function block	Description
1	IL_ModbusTCPReadCoils, page 221	Reads a number of bits from "BitData" starting from the initial address
2	IL_ModbusTCPReadDiscreteInputs, page 224	Reads a number of bits from "BitData" starting from the initial address
3	IL_ModbusTCPReadHoldingRegisters, page 226	Reads a number of registers from "RegisterData" starting from the initial address
4	IL_ModbusTCPReadInputRegisters, page 228	Reads a number of registers from "RegisterData" starting from the initial address
5	IL_ModbusTCPWriteSingleCoil, page 230	Writes a bit into "BitData" starting from the initial address
6	IL_ModbusTCPWriteSingleRegister, page 232	Writes a register into "RegisterData" starting from the initial address
15	IL_ModbusTCPWriteMultipleCoils, page 234	Writes a number of bits into "BitData" starting from the initial address
16	IL_ModbusTCPWriteMultipleRegisters, page 236	Writes a number of registers into "RegisterData" starting from the initial address
23	IL_ModbusTCPReadWriteMultipleRegisters, page 238	Executes a combined read/write operation on "RegisterData"
43/14	IL_ModbusTCPReadDeviceIdentification, page 241	Reads the default device information of the Modbus TCP server

Tab. 5-21: Supported Modbus function codes

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#0208	16#0000	The "Address" input is invalid.
INPUT_INVALID_ERROR	16#0204	16#0000	The "Port" input is invalid.
OTHER_ERROR	16#0040	16#0000	Could not open socket
OTHER_ERROR	16#0041	16#0000	Could not set I/O mode of socket
OTHER_ERROR	16#0045	16#0209	Remote host denied connection establishment - No ModbusTCP server is running there
OTHER_ERROR	16#0045	16#020A	Remote host does not respond.
OTHER_ERROR	16#0045	16#020B	Remote host not found.
OTHER_ERROR	16#0045	16#020C	No route to remote host.
COMMUNICATION_ERROR	16#020E	16#0000	The connection was closed.
COMMUNICATION_ERROR	16#0211	16#0000	Remote host closed the connection.

Tab. 5-22: Error codes of the IL_ModbusTCPClient

5.3.3 IL_ModbusTCPReadCoils

Brief description The IL_ModbusTCPReadCoils function block cyclically reads multiple coils from a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-23: Library assignment

Interface description

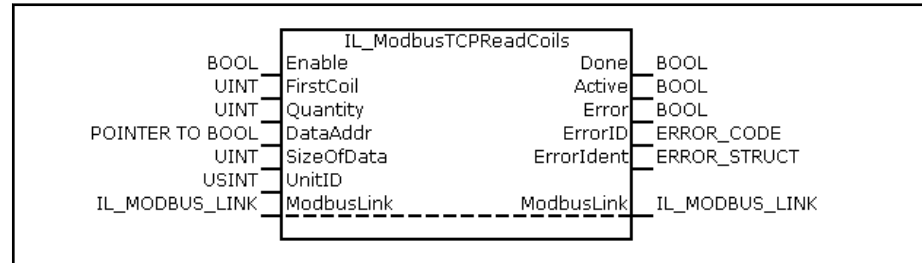


Fig. 5-18: IL_ModbusTCPReadCoils function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	FirstCoil	UINT	First coil to be read from
	Quantity	UINT	Number of coils to be read
	DataAddr	POINTER TO BOOL	Address of the data range the coils are to be copied to
	SizeOfData	UINT	Size of the data range created in "DataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed. Received data was transferred to the range created in "DataAddr"
	Active	BOOL	Transfer running
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 223 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-24: IL_ModbusTCPReadCoils interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
FirstCoil	UINT	0	65535	0	Rising edge at "Enable"
Quantity	UINT	1	2000	0	Rising edge at "Enable"
DataAddr	POINTER TO BOOL	0		0	Rising edge at "Enable"

Name	Type	Min. value	Max. value	Default value	Effective
SizeOfData	UINT	1	2000	0	Rising edge at "Enable"
UnitID	USINT	0	255	255	Rising edge at "Enable"

Tab. 5-25: Input behavior IL_ModbusTCPClient

Functional Description The IL_ModbusTCPReadCoils function block implements the Modbus function code 1.

It cyclically reads multiple coils from a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Enable, Done At a rising edge at "Enable", the function block checks its inputs. Subsequently, it cyclically sends a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set for one PLC cycle. At a falling edge at "Enable", the function block is disabled. If it is still transferred, processing continues using the IL_ModbusTCPClient function block and the result is discarded.

Error, ErrorID, ErrorIdent If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.

FirstCoil, Quantity Modbus provides 65536 readable and writable coils. Each coil can either be set or deleted. The "FirstCoil" input specifies the first coil to be read. From this coil, "Quantity" coils are read. Up to 2000 coils can be read per call.

DataAddr, SizeOfData The received data is stored in the data range specified in "DataAddr". An array of Bool is expected as reception buffer. The array size specified in "SizeOfData" has to be at least equal to the number of coils to be read. It is recommended to use the SIZEOF() operator to determine the array size.

UnitID The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.

ModbusLink The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.

Troubleshooting The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#00001811	16#00000000	The address specified at the "DataAddr" input is 0.
INPUT_INVALID_ERROR	16#00001812	16#00000000	The "SizeOfData" input is 0
INPUT_INVALID_ERROR	16#00001813	16#00000000	The "Quantity" input is 0
INPUT_RANGE_ERROR	16#00001814	16#00000000	The "Quantity" input is higher than 2000

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00001815	16#00000000	The value at the "Quantity" input is higher than "SizeOf-Data"
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-26: Error codes of the IL_ModbusTCPReadCoils

5.3.4 IL_ModbusTCPReadDiscreteInputs

Brief description The IL_ModbusTCPReadDiscreteInputs function block cyclically reads multiple discrete inputs from a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-27: Library assignment

Interface description

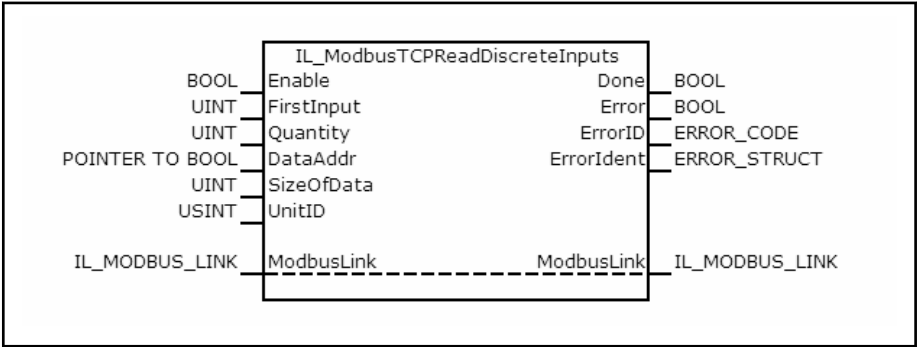


Fig. 5-19: IL_ModbusTCPReadDiscreteInputs function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	FirstCoil	UINT	First discrete input to be read from
	Quantity	UINT	Number of inputs to be read
	DataAddr	POINTER TO BOOL	Address of the data range to copy the inputs to
	SizeOfData	UINT	Size of the data range created in "DataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed. Received data was transferred to the range created in "DataAddr"
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 226 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-28: IL_ModbusTCPReadDiscreteInputs interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
FirstInput	UINT	0	65535	0	Rising edge at "Enable"
Quantity	UINT	1	2000	0	Rising edge at "Enable"
DataAddr	POINTER TO BOOL	0		0	Rising edge at "Enable"
SizeOfData	UINT	1	2000	0	Rising edge at "Enable"
UnitID	USINT	0	255	255	Rising edge at "Enable"

Tab. 5-29: Input behavior of the IL_ModbusTCPReadDiscreteInputs

Functional Description

The IL_ModbusTCPReadDiscreteInputs function block implements the Modbus function code 2.

It cyclically reads multiple discrete inputs from a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Enable, Done

At a rising edge at "Enable", the function block checks its inputs. Subsequently, it cyclically sends a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set for one PLC cycle. At a falling edge at "Enable", the function block is disabled. If it is still transferred, processing continues using the IL_ModbusTCPClient function block and the result is discarded.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.

FirstInput, Quantity

Modbus provides 65536 read-only discrete inputs. Each input can either be set or deleted. The "FirstInput" input specifies the first input to be read. From this input, "Quantity" inputs are read. Up to 2000 inputs can be read per call.

- DataAddr, SizeOfData** The received data is stored in the data range specified in "DataAddr". An array of Bool is expected as reception buffer. The array size specified in "SizeOfData" has to be at least equal to the number of inputs to be read. It is recommended to use the SIZEOF() operator to determine the array size.
- UnitID** The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
- ModBusLink** The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
- Troubleshooting** The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#00001811	16#00000000	The address specified at the "DataAddr" input is 0.
INPUT_INVALID_ERROR	16#00001812	16#00000000	The "SizeOfData" input is 0
INPUT_INVALID_ERROR	16#00001813	16#00000000	The "Quantity" input is 0
INPUT_RANGE_ERROR	16#00001814	16#00000000	The "Quantity" input is higher than 2000
INPUT_INVALID_ERROR	16#00001815	16#00000000	The value at the "Quantity" input is higher than "SizeOfData"
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-30: Error codes of the IL_ModbusTCPReadDiscreteInputs

5.3.5 IL_ModbusTCPReadHoldingRegisters

- Brief description** The IL_ModbusTCPReadHoldingRegisters function block cyclically reads multiple holding registers from a ModbusTCP server.
- Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTCP/client

Tab. 5-31: Library assignment

Interface description

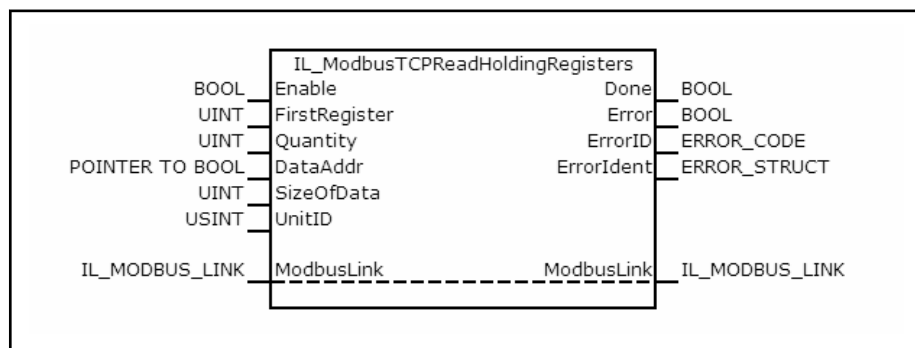


Fig. 5-20: IL_ModbusTCPReadHoldingRegisters function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	FirstRegister	UINT	First register to be read from
	Quantity	UINT	Number of registers to be read
	DataAddr	POINTER TO WORD	Address of the data range to copy the registers to
	SizeOfData	UINT	Size of the data range created in "DataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed. Received data was transferred to the range created in "DataAddr"
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 228 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-32: IL_ModbusTCPReadHoldingRegisters interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
FirstRegister	UINT	0	65535	0	Rising edge at "Enable"
Quantity	UINT	1	125	0	Rising edge at "Enable"
DataAddr	POINTER TO WORD	0		0	Rising edge at "Enable"
SizeOfData	UINT	2	250	0	Rising edge at "Enable"
UnitID	USINT	0	255	255	Rising edge at "Enable"

Tab. 5-33: Input behavior of the IL_ModbusTCPReadHoldingRegisters

Functional Description

The IL_ModbusTCPReadHoldingRegisters function block implements the Modbus function code 3. It cyclically reads multiple holding registers from a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Enable, Done

At a rising edge at "Enable", the function block checks its inputs. Subsequently, it cyclically sends a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set for one PLC cycle. At a falling edge at "Enable", the function block is disabled. If it is still transferred, processing continues using the IL_ModbusTCPClient function block and the result is discarded.

- Error, ErrorID, ErrorIdent** If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.
- The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.
- FirstRegister, Quantity** Modbus provides 65536 readable and writable holding registers. Each register has a width of one word (2 bytes). It can assume a value between 0 and 65535. The "FirstRegister" input specifies the first register to be read. From this register, "Quantity" registers are read. Up to 125 registers can be read per call.
- DataAddr, SizeOfData** The received data is stored in the data range specified in "DataAddr". An array of Word is expected as reception buffer. The array size specified in "SizeOfData" has to be at least double the number of registers to be read. It is recommended to use the SIZEOF() operator to determine the array size.
- UnitID** The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
- ModBusLink** The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
- Troubleshooting** The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#00001811	16#00000000	The address specified at the "DataAddr" input is 0.
INPUT_INVALID_ERROR	16#00001812	16#00000000	The "SizeOfData" input is 0
INPUT_INVALID_ERROR	16#00001813	16#00000000	The "Quantity" input is 0
INPUT_RANGE_ERROR	16#00001814	16#00000001	The "Quantity" input is higher than 125
INPUT_INVALID_ERROR	16#00001816	16#00000000	The value at the "Quantity"*2 input is higher than "SizeOfData"
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-34: Error codes of the IL_ModbusTCPReadHoldingRegisters

5.3.6 IL_ModbusTCPReadInputRegisters

- Brief description** The IL_ModbusTCPReadInputRegisters function block cyclically reads multiple input registers from a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-35: Library assignment

Interface description

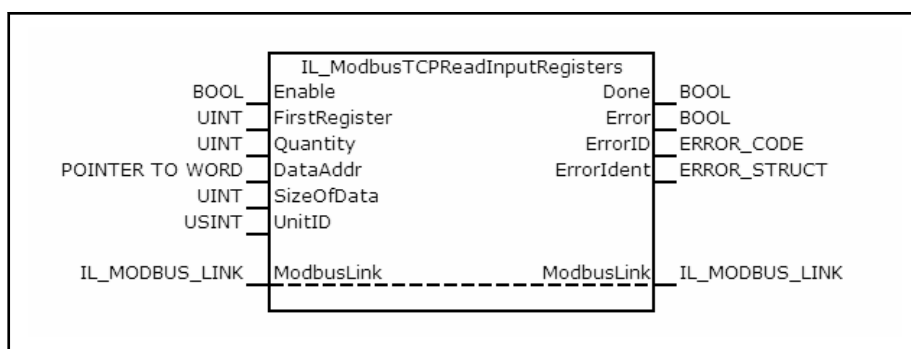


Fig. 5-21: IL_ModbusTCPReadInputRegisters function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	FirstRegister	UINT	First register to be read from
	Quantity	UINT	Number of registers to be read
	DataAddr	POINTER TO WORD	Address of the data range to copy the registers to
	SizeOfData	UINT	Size of the data range created in "DataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed. Received data was transferred to the range created in "DataAddr"
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 230 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-36: IL_ModbusTCPReadInputRegisters interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
FirstRegister	UINT	0	65535	0	Rising edge at "Enable"
Quantity	UINT	1	125	0	Rising edge at "Enable"
DataAddr	POINTER TO WORD	0		0	Rising edge at "Enable"
SizeOfData	UINT	2	250	0	Rising edge at "Enable"
UnitID	USINT	0	255	255	Rising edge at "Enable"

Tab. 5-37: Input behavior of the IL_ModbusTCPReadInputRegisters

Functional Description	The IL_ModbusTCPReadInputRegisters function block implements the Modbus function code 4. It cyclically reads multiple input registers from a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.
Enable, Done	At a rising edge at "Enable", the function block checks its inputs. Subsequently, it cyclically sends a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set for one PLC cycle. At a falling edge at "Enable", the function block is disabled. If it is still transferred, processing continues using the IL_ModbusTCPClient function block and the result is discarded.
Error, ErrorID, ErrorIdent	If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset. The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.
FirstRegister, Quantity	Modbus provides 65536 read-only input registers. Each register has a width of one word (2 bytes). It can assume a value between 0 and 65535. The "FirstRegister" input specifies the first register to be read. From this register, "Quantity" registers are read. Up to 125 registers can be read per call.
DataAddr, SizeOfData	The received data is stored in the data range specified in "DataAddr". An array of Word is expected as reception buffer. The array size specified in "SizeOfData" has to be at least double the number of registers to be read. It is recommended to use the SIZEOF() operator to determine the array size.
UnitID	The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
ModBusLink	The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
Troubleshooting	The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#00001811	16#00000000	The address specified at the "DataAddr" input is 0.
INPUT_INVALID_ERROR	16#00001812	16#00000000	The "SizeOfData" input is 0
INPUT_INVALID_ERROR	16#00001813	16#00000000	The "Quantity" input is 0
INPUT_RANGE_ERROR	16#00001814	16#00000001	The "Quantity" input is higher than 125
INPUT_INVALID_ERROR	16#00001816	16#00000000	The value at the "Quantity"*2 input is higher than "SizeOfData"
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-38: Error codes of the IL_ModbusTCPReadInputRegisters

5.3.7 IL_ModBusTCPWriteSingleCoil

Brief description	The IL_ModBusTCPWriteSingleCoil function block writes a coil once to a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS	RIL_ModbusTcp/client
Basic function package PLC	

Tab. 5-39: Library assignment

Interface description

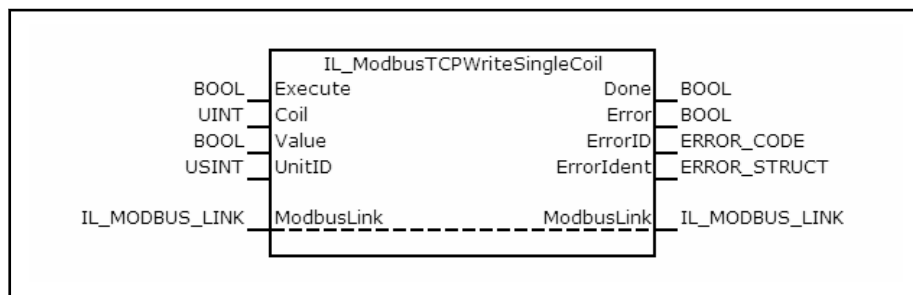


Fig. 5-22: IL_ModBusTCPWriteSingleCoil function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Processing the function block enabled (edge-controlled).
	Coil	UINT	Coil to be written
	Value	BOOL	Value to be written
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 232 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-40: IL_ModBusTCPWriteSingleCoil interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Coil	UINT	0	65535	0	Rising edge at "Execute"
Value	BOOL			FALSE	Rising edge at "Execute"
UnitID	USINT	0	255	255	Rising edge at "Execute"

Tab. 5-41: Input behavior of the IL_ModBusTCPWriteSingleCoil

Functional Description

The IL_ModBusTCPWriteSingleCoil function block implements the Modbus function code 5. It writes a coil once to the ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Execute, Done	At a rising edge at "Execute", the function block checks its inputs. Subsequently, it sends once a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set. At a falling edge at "Execute", the function block continues running until the current transfer is completed. "Done" is then set for one PLC cycle.
Error, ErrorID, ErrorIdent	If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset. The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.
Coil, Value	Modbus provides 65536 readable and writable coils. Each coil can either be set or deleted. The "Coil" input specifies the coil to be written. The value in "Value" is written to this coil.
UnitID	The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
ModBusLink	The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
Troubleshooting	The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-42: Error codes of the IL_ModBusTCPWriteSingleCoil

5.3.8 IL_ModBusTCPWriteSingleRegister

Brief description	The IL_ModBusTCPWriteSingleRegister function block writes a holding register once to a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS	RIL_ModbusTcp/client
Basic function package PLC	

Tab. 5-43: Library assignment

Interface description

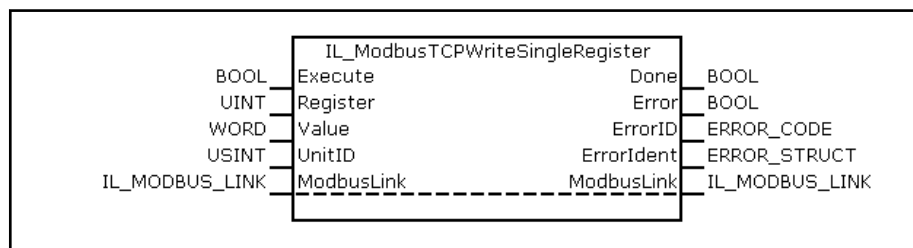


Fig. 5-23: IL_ModBusTCPWriteSingleRegister function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Processing the function block enabled (edge-controlled).
	Register	UINT	Register to be written
	Value	WORD	Value to be written
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 234 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-44: IL_ModbusTCPWriteSingleRegister interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
HoldingRegister	UINT	0	65535	0	Rising edge at "Execute"
Value	WORD	0	65535	0	Rising edge at "Execute"
UnitID	USINT	0	255	255	Rising edge at "Execute"

Tab. 5-45: Input behavior of the IL_ModbusTCPWriteSingleRegister

Functional Description

The IL_ModbusTCPWriteSingleRegister function block implements the Modbus function code 6. It reads and writes one holding register to a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Execute, Done

At a rising edge at "Execute", the function block checks its inputs. Subsequently, it sends once a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set. At a falling edge at "Execute", the function block continues running until the current transfer is completed. "Done" is then set for one PLC cycle.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.

Register, Value

Modbus provides 65536 readable and writable holding registers. Each register has a width of one word (2 bytes). It can assume a value between 0 and 65535. The "Register" input specifies the register to be written. The value at the "Value" input is written to this register.

- UnitID** The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
- ModBusLink** The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
- Troubleshooting** The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-46: Error codes of the IL_ModBusTCPWriteSingleRegister

5.3.9 IL_ModBusTCPWriteMultipleCoils

Brief description The IL_ModBusTCPWriteMultipleCoils function block writes multiple coils once to a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-47: Library assignment

Interface description

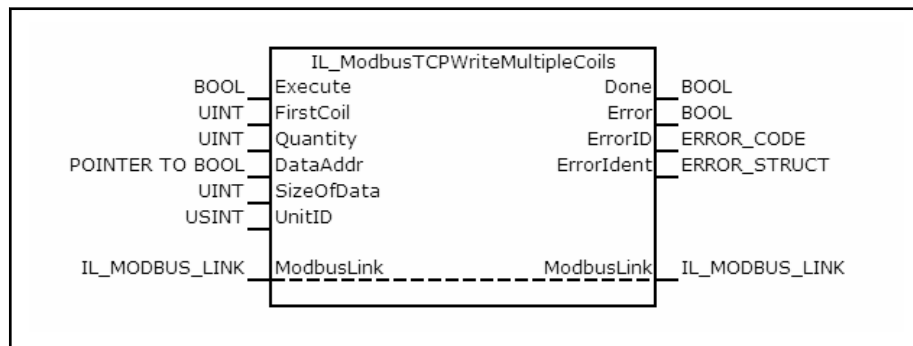


Fig. 5-24: IL_ModBusTCPWriteMultipleCoils function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Processing the function block enabled (edge-controlled).
	FirstCoil	UINT	First coil to be written from
	Quantity	UINT	Number of coils to be written
	DataAddr	POINTER TO BOOL	Address of the data range the coils are to be read from
	SizeOfData	UINT	Size of the data range created in "DataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 236 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-48: IL_ModbusTCPWriteMultipleCoils interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
FirstCoil	UINT	0	65535	0	Rising edge at "Execute"
Quantity	UINT	1	2000	0	Rising edge at "Execute"
DataAddr	POINTER TO BOOL	0		0	Rising edge at "Execute"
SizeOfData	UINT	1	2000	0	Rising edge at "Execute"
UnitID	USINT	0	255	255	Rising edge at "Execute"

Tab. 5-49: Input behavior of the IL_ModbusTCPWriteMultipleCoils

Functional Description

The IL_ModbusTCPWriteMultipleCoils function block implements the Modbus function code 15.

It writes multiple coils once to the ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Execute, Done

At a rising edge at "Execute", the function block checks its inputs. Subsequently, it sends once a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set. At a falling edge at "Execute", the function block continues running until the current transfer is completed. "Done" is then set for one PLC cycle.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.

FirstCoil, Quantity

Modbus provides 65536 readable and writable coils. Each coil can either be set or deleted. The "FirstCoil" input specifies the first coil to be written. From this coil, "Quantity" coils are written. Up to 2000 coils can be written per call.

DataAddr, SizeOfData

The data to be transmitted is retrieved from the data range specified in "DataAddr". An array of Bool is expected as transmission buffer. The array size

specified in "SizeOfData" has to be at least equal to the number of coils to be written. It is recommended to use the SIZEOF() operator to determine the array size.

UnitID The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.

ModbusLink The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.

Troubleshooting The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#00001811	16#00000000	The address specified at the "DataAddr" input is 0.
INPUT_INVALID_ERROR	16#00001812	16#00000000	The "SizeOfData" input is 0
INPUT_INVALID_ERROR	16#00001813	16#00000000	The "Quantity" input is 0
INPUT_RANGE_ERROR	16#00001814	16#00000000	The "Quantity" input is higher than 2000
INPUT_INVALID_ERROR	16#00001815	16#00000000	The value at the "Quantity" input is higher than "SizeOfData"
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-50: Error codes of the IL_ModbusTCPWriteMultipleCoils

5.3.10 IL_ModbusTCPWriteMultipleRegisters

Brief description The IL_ModbusTCPWriteMultipleRegisters function block writes multiple holding registers once to a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-51: Library assignment

Interface description

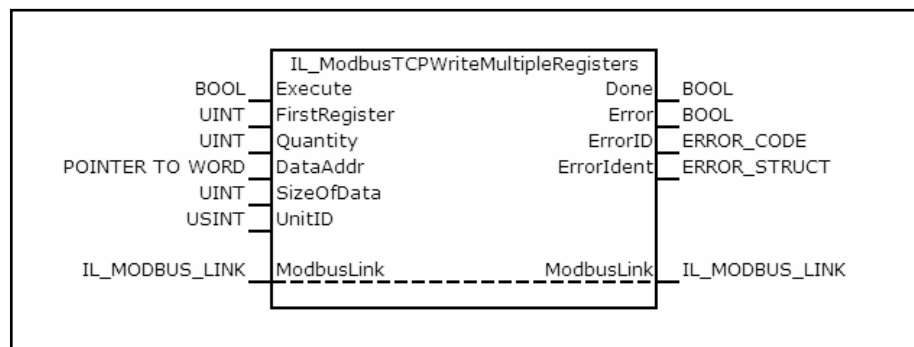


Fig. 5-25: IL_ModbusTCPWriteMultipleRegisters function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Processing the function block enabled (edge-controlled).
	FirstRegister	UINT	First register to be written from
	Quantity	UINT	Number of registers to be written
	DataAddr	POINTER TO WORD	Address of the data range the registers are to be copied from
	SizeOfData	UINT	Size of the data range created in "DataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed.
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 238 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-52: IL_ModBusTCPWriteMultipleRegisters interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
FirstRegister	UINT	0	65535	0	Rising edge at "Execute"
Quantity	UINT	1	125	0	Rising edge at "Execute"
DataAddr	POINTER TO WORD	0		0	Rising edge at "Execute"
SizeOfData	UINT	2	250	0	Rising edge at "Execute"
UnitID	USINT	0	255	255	Rising edge at "Execute"

Tab. 5-53: Input behavior of the IL_ModBusTCPWriteMultipleRegisters

Functional Description

The IL_ModBusTCPWriteMultipleRegisters function block implements the Modbus function code 16. It reads and writes multiple holding registers to a ModbusTCP server. Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Execute, Done

At a rising edge at "Execute", the function block checks its inputs. Subsequently, it sends once a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set. At a falling edge at "Execute", the function block continues running until the current transfer is completed. "Done" is then set for one PLC cycle.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function

block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.

FirstRegister, Quantity Modbus provides 65536 readable and writable holding registers. Each register has a width of one word (2 bytes). It can assume a value between 0 and 65535. The "FirstRegister" input specifies the first register to be written. From this register, "Quantity" registers are written. Up to 125 registers can be written per call.

DataAddr, SizeOfData The data to be transmitted is retrieved from the data range specified in "DataAddr". An array of Word is expected as transmission buffer. The array size specified in "SizeOfData" has to be at least double the size of the number of registers to be written. It is recommended to use the SIZEOF() operator to determine the array size.

UnitID The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.

ModBusLink The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.

Troubleshooting The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#00001811	16#00000000	The address specified at the "DataAddr" input is 0.
INPUT_INVALID_ERROR	16#00001812	16#00000000	The "SizeOfData" input is 0
INPUT_INVALID_ERROR	16#00001813	16#00000000	The "Quantity" input is 0
INPUT_RANGE_ERROR	16#00001814	16#00000001	The "Quantity" input is higher than 125
INPUT_INVALID_ERROR	16#00001816	16#00000000	The value at the "Quantity"*2 input is higher than "SizeOfData"
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-54: Error codes of the IL_ModBusTCPWriteMultipleRegisters

5.3.11 IL_ModBusTCPReadWriteMultipleRegisters

Brief description The IL_ModBusTCPReadWriteMultipleRegisters function block cyclically reads and writes multiple holding registers from a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-55: Library assignment

Interface description

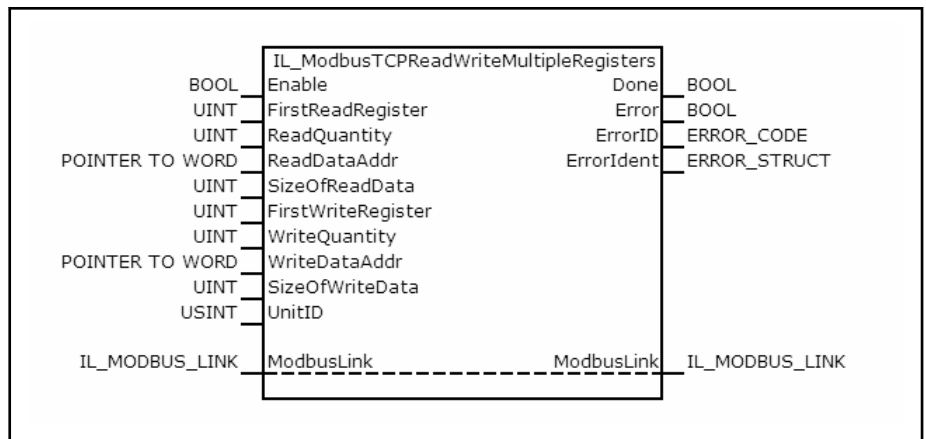


Fig. 5-26: IL_ModbusTCPReadWriteMultipleRegisters function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Function block processing enabled (level-controlled).
	FirstReadRegister	UINT	First register to be read from
	ReadQuantity	UINT	Number of registers to be read
	ReadDataAddr	POINTER TO WORD	Address of the data range to copy the registers to
	SizeOfReadData	UINT	Size of the data range created in "ReadDataAddr" in bytes
	FirstWriteRegister	UINT	First register to be written from
	WriteQuantity	UINT	Number of registers to be written
	WriteDataAddr	POINTER TO WORD	Address of the data range the registers are to be copied from
	SizeOfWriteData	UINT	Size of the data range created in "WriteDataAddr" in bytes
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed. Received data was transferred to the range created in "DataAddr"
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information See also "Troubleshooting" on page 241 .
VAR_IN_OUT	ModbusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-56: IL_ModbusTCPReadWriteMultipleRegisters interface

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
FirstReadRegister	UINT	0	65535	0	Rising edge at "Enable"

Name	Type	Min. value	Max. value	Default value	Effective
ReadQuantity	UINT	1	125	0	Rising edge at "Enable"
ReadDataAddr	POINTER TO WORD	0		0	Rising edge at "Enable"
SizeOfReadData	UINT	2	250	0	Rising edge at "Enable"
FirstWriteRegister	UINT	0	65535	0	Rising edge at "Enable"
WriteQuantity	UINT	1	125	0	Rising edge at "Enable"
WriteDataAddr	POINTER TO WORD	0		0	Rising edge at "Enable"
SizeOfWriteData	UINT	2	250	0	Rising edge at "Enable"
UnitID	USINT	0	255	255	Rising edge at "Enable"

Tab. 5-57: Input behavior of the IL_ModBusTCPReadWriteMultipleRegisters

Functional description

The IL_ModBusTCPReadWriteMultipleRegisters function block implements the Modbus function code 23.

It cyclically reads and writes multiple holding registers from a ModbusTCP server.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Enable, Done

At a rising edge at "Enable", the function block checks its inputs. Subsequently, it cyclically sends a request to the Modbus server via the IL_ModbusTCPClient specified at "ModbusLink" and waits for response. After a successful transfer cycle, "Done" is set for one PLC cycle. At a falling edge at "Enable", the function block is disabled. If it is still transferred, processing continues using the IL_ModbusTCPClient function block and the result is discarded.

Error, ErrorID, ErrorIdent

If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Error" output is reset.

The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.

FirstReadRegister, ReadQuantity

Modbus provides 66536 readable and writable holding registers. Each register has a width of one word (2 bytes). It can assume a value between 0 and 65535. The "FirstReadRegister" input specifies the first register to be read. From this register, "ReadQuantity" registers are read. Up to 125 registers can be read per call.

FirstWriteRegister, WriteQuantity

Modbus provides 66536 readable and writable holding registers. Each register has a width of one word (2 bytes). It can assume a value between 0 and 65535. The "FirstWriteRegister" input specifies the first register to be written. From this register, "WriteQuantity" registers are written. Up to 125 registers can be written per call.

ReadDataAddr, SizeOfReadData

The received data is stored in the data range specified in "ReadDataAddr". An array of Word is expected as reception buffer. The array size specified in

	"SizeOfReadData" has to be at least double the size of the number of registers to be read. It is recommended to use the SIZEOF() operator to determine the array size.
WriteDataAddr, SizeOfWriteData	The data to be transmitted is retrieved from the data range specified in "ReadDataAddr". An array of Word is expected as transmission buffer. The array size specified in "SizeOfWriteData" has to be at least double the size of the number of registers to be written. It is recommended to use the SIZEOF() operator to determine the array size.
UnitID	The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
ModBusLink	The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
Troubleshooting	The function block uses the error table F_RELATED_TABLE, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#1810	16#0000	The link specified at the "ModbusLink" input is invalid.
INPUT_INVALID_ERROR	16#1811	16#0000	The address specified at the "ReadDataAddr" input is 0
INPUT_INVALID_ERROR	16#1811	16#0001	The address specified at the "WriteDataAddr" input is 0
INPUT_INVALID_ERROR	16#1812	16#0000	The "SizeOfReadData" input is 0
INPUT_INVALID_ERROR	16#1812	16#0001	The "SizeOfWriteData" input is 0
INPUT_INVALID_ERROR	16#1813	16#0000	The "ReadQuantity" input is 0
INPUT_INVALID_ERROR	16#1813	16#0001	The "WriteQuantity" input is 0
INPUT_RANGE_ERROR	16#1814	16#0001	The "ReadQuantity" input is bigger than 125
INPUT_RANGE_ERROR	16#1814	16#0002	The "WriteQuantity" input is bigger than 125
INPUT_INVALID_ERROR	16#1816	16#0000	The value at the "ReadQuantity"*2 input is bigger than "SizeOfReadData"
INPUT_INVALID_ERROR	16#1816	16#0001	The value at the "WriteQuantity"*2 input is bigger than "SizeOfWriteData"
RESOURCE_ERROR	16#1817	Modbus error code	The Modbus operation returns an error code

Tab. 5-58: Error codes of the IL_ModbusTCPReadWriteMultipleRegisters

5.3.12 IL_ModbusTCPReadDeviceIdentification

Brief description The IL_ModbusTCPReadDeviceIdentification function block reads all available device information on the Modbus client device and provides them at its outputs.

Thus, a connection managed by the IL_ModbusTCPClient function block is required.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_ModbusTcp/client

Tab. 5-59: Library assignment

Interface description

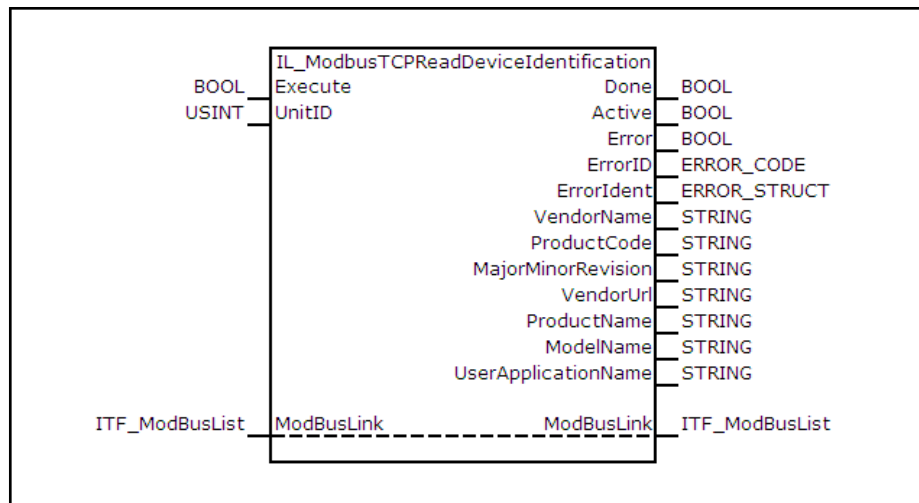


Fig. 5-27: IL_ModbusTCPReadDeviceIdentification function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Processing the function block enabled (edge-controlled).
	UnitID	USINT	Unit ID or slave ID of the client
VAR_OUTPUT	Done	BOOL	One transfer cycle is completed. Received data was transferred to the range created in "DataAddr"
	Active	BOOL	Transfer running
	Error	BOOL	Error while processing the function block.
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification.
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, it contains detailed error information. Also refer to "Troubleshooting" on page 243 .
	VendorName	STRING	Vendor name
	ProductCode	STRING	Product code
	MajorMinorRevision	STRING	Version number
	VendorUrl	STRING	Web address of the vendor
	ProductName	STRING	Product name
	ModelName	STRING	Model name
	UserApplicationName	STRING	Application name
VAR_IN_OUT	ModBusLink	IL_MODBUS_LINK	Link to existing Modbus connection is provided by the IL_ModbusTCPClient function block

Tab. 5-60: IL_ModbusTCPReadDeviceIdentification interface

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
UnitID	USINT	0	255	255	Rising edge at "Execute"

Tab. 5-61: Input behavior of the IL_ModbusTCPReadDeviceIdentification

Functional Description	The IL_ModbusTCPReadDeviceIdentification function block reads all available device information on the Modbus client device and provides them at its outputs.
Execute, Active, Done	At a rising edge at "Execute", the function block checks its inputs. Subsequently, it sends the request to the Modbus server via the IL_ModbusTCP-Client specified at "ModbusLink" and waits for response. During this period, the "Active" output is set. After a successful transfer reception, "Done" is set. At a falling edge at "Execute", the function block continues running until no data is received anymore or an error occurs.
Error, ErrorID, ErrorIdent	If an error occurs while processing the function block, the output "Error" is set. "ErrorID" and "ErrorIdent" contain detailed error information. Function block processing is paused. At a falling edge at "Enable", the "Execute" output is reset. The error outputs are set if errors occur at the function block inputs and in the ModbusTCP protocol.
UnitID	The unit ID of the ModbusTCP protocol corresponds to the slave ID of the Modbus RTU. It is used to differ between different clients on the bus or to access different address ranges.
ModbusLink	The "ModbusLink" input references the actual ModbusTCP connection. It is output by the IL_ModbusTCPClient function block and valid when the connection is established.
Troubleshooting	The function block uses the F_RELATED_TABLE, 16#00000170 error table. It can generate the following error messages in Additional1 and Additional2:

ErrorID	Additional1	Additional2	Description
RESOURCE_ERROR	16#00001810	16#00000000	The link specified at the "ModbusLink" input is invalid.
RESOURCE_ERROR	16#00001817	Modbus error code	The Modbus operation returns an error code

Tab. 5-62: Error codes of the IL_ModbusTCPReadDeviceIdentification

6 RIL_SocketComm.library

6.1 Introduction and overview

The RIL_SocketComm library contains functions and function blocks to establish a connection to and data transfer via TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).

TCP and UDP are IP-based internet data transport protocols.

While TCP is a connection-oriented protocol that establishes connections and maintains them while the application transfers data, UDP is a package-oriented connection-free protocol that sends data packages within a network from one application to another.

RIL_SocketComm supports synchronous and asynchronous communication.

The synchronous communication is supported by functions.

Function blocks are used for the asynchronous communication. It is not recommended to use functions and function blocks within the same TCP or UDP connection.



The IP-based communication is not real-time capable. That means that it cannot be guaranteed that data reaches its target within a specific time slot. Thus, it should only be used for pure data exchange.

It is not to be used for process synchronization and controlling time-critical inputs/outputs.

The RIL_SocketComm library uses SysSockets library functions. The IP-based communication is only executed by the operating system of the control. Thus, there can be differences in the runtime behavior on different controls.



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

6.2 I/O time response of the function blocks

Every function block supports the common error structure and the defined behavior with the outputs "Error", "ErrorID" and "ErrorIdent".

All function blocks with the "Enable" input and "Done" output have the same level-controlled runtime behavior. A rising edge at the "Enable" input starts the execution. A falling edge stops the execution and resets all binary outputs.

If "Done" is TRUE, the function block has successfully completed its task. The function block remains in this state until "Enable" is reset. No further actions are performed.

If there is an error, "Error" is set and "ErrorID" and "ErrorIdent" contain the error code. "Done" and "Error" are reset when "Enable" is reset.

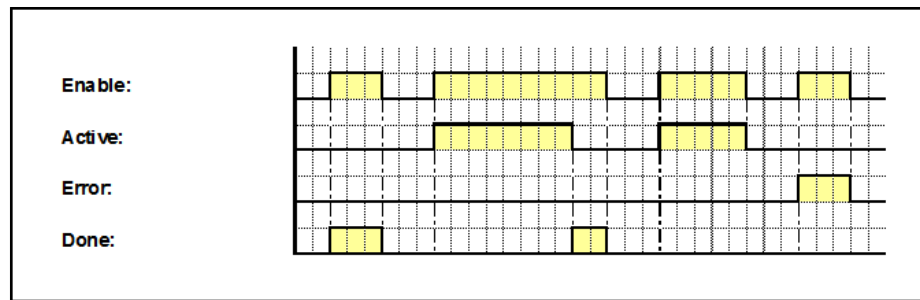


Fig. 6-1: Time diagram for Enable, Active, Error and Done

6.3 Functional description

6.3.1 TCP

TCP is a connection-orientated IP protocol. Before data is exchanged, the client has to be connected to an open TCP server port. The data flow is based on characters. The sent data is subdivided into data packages by the TCP protocol, transferred and recombined into a character stream at the other side (server or client).

The order of the characters are sent is always maintained. If an error occurs during transmission, the package in question is automatically resent. Therefore, data cannot be lost. A connection loss is reported in an error code at next read or write access.

TCP is suitable for non-cyclic communication, when data has to reliably reach the receiver. A good example is a log file. It is not important when the data reaches its target but it has to reach the remote host.

The following chart shows a TCP connection with subsequent data transmission via TCP.

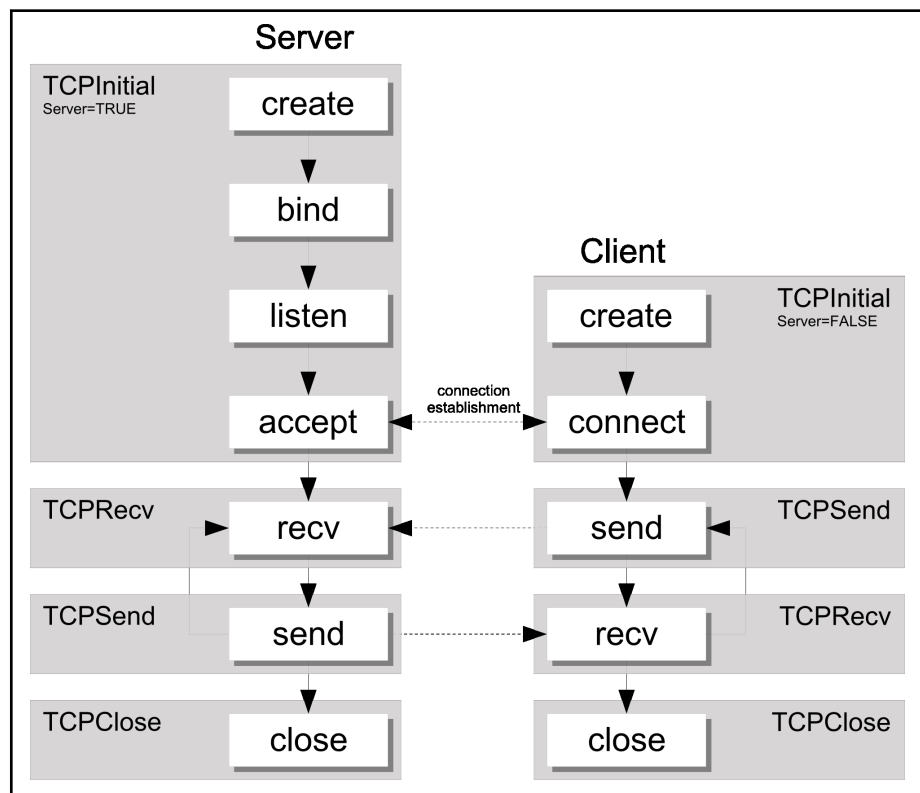


Fig. 6-2: TCP server client flow chart

Asynchronous function blocks for a TCP connection

- [IL_TCPConnectionAsync on page 259](#)
- [IL_TCPInitialAsync on page 263](#)
- [IL_TCPSendAsync on page 267](#)
- [IL_TCPRecvAsync on page 268](#)
- [IL_TCPRecvPacketAsync on page 270](#)
- [IL_TCPCloseAsync on page 272](#)

Synchronous functions for a TCP connection

- [IL_TCPInitial on page 274](#)
- [IL_TCPSend on page 276](#)
- [IL_TCPRecv on page 277](#)
- [IL_TCPRecvPacket on page 278](#)
- [IL_TCPClose on page 279](#)

Details on implementation:

- [TCP asynchronous function blocks on page 258](#)
- [TCP synchronous functions on page 274](#)

TCP server example

The following example shows a simple TCP server application, which sends an ASCII text using the RIL_SocketComm library.

When started, the program opens a TCP server on port 4242 and waits for an incoming connection. After the connection has been established, the text is sent and the connection is closed. The program opens the server port again and waits for further incoming connections.

Simple TCP server program (declaration)

```
(* This very simple test program waits for an incoming TCP connection on port 4242. *)
(* After the connection is established, it sends the string "Hello World!"          *)
(* and closes the connection                                                         *)
```

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    (* used function block instances *)
    fbTCPConnectionAsync: IL_TCPConnectionAsync;
    fbTCPSendAsync: IL_TCPSendAsync;
```

```
    (* variables *)
    udiSocket: UDINT; (* socket handle *)
    strHello: STRING := 'Hello World!$r$n'; (* text to send *)
```

```
END_VAR
```

Simple TCP server program (implementation)

```
(* Check if connection is established *)
IF fbTCPConnectionAsync.Done = FALSE THEN
    (* No connection established yet => setup server mode and wait *)
    fbTCPConnectionAsync(Enable:=TRUE, Server:=TRUE, Port:=4242, Socket=>udiSocket);
    IF fbTCPConnectionAsync.Error = TRUE THEN
        (* error while establishing connection => reset everything *)
        fbTCPConnectionAsync(Enable:=FALSE, Socket=>udiSocket);
    END_IF
ELSE
    (* connection established => send text *)
    fbTCPSendAsync(Enable:=TRUE, Socket:=udiSocket, ValueAdr:=ADR(strHello), NoOfBytes:=LEN(strHello));
    IF fbTCPSendAsync.Done=TRUE OR fbTCPSendAsync.Error=TRUE THEN
        (* text completely send or some error => close connection *)
        fbTCPSendAsync(Enable:=FALSE);
        fbTCPConnectionAsync(Enable:=FALSE, Socket=>udiSocket);
    END_IF
END_IF
```

Use the Windows "Telnet" program to connect to the control. The first parameter is the IP address, the second parameter is the port:

RIL_SocketComm.library

Telnet 192.168.1.1 4242

TCP client example

The following example shows a TCP client, which connects to a TCP server at port 4242. After the connection has been established, the string sends "Hello" and waits for a response. Once the response package is received, the connection is closed.

Simple TCP client program (declaration)

```
PROGRAM PLC_PRG
VAR
    fbConnection: IL_TCPConnectionAsync;
    fbSend: IL_TCPSendAsync;
    fbRecv: IL_TCPRecvAsync;

    udiSocket: UDINT := SOCKET_INVALID;

    strSend: STRING := 'Hello';
    strRecv: STRING;

    iState: INT := 0;
END_VAR
```

Simple TCP client program (implementation)

```
CASE iState OF

0:
    fbConnection(Enable:=FALSE);
    fbSend(Enable:=FALSE);
    fbRecv(Enable:=FALSE);
    diSocket := SOCKET_INVALID;
    iState := 1;

1:
    fbConnection(Enable:=TRUE, Server:=FALSE, Address:='192.168.42.254', Port:=4242);
    IF fbConnection.Done=TRUE THEN
        (* Connection established *)
        udiSocket := fbConnection.Socket;
        iState := 2;
    END_IF

2:
    (* Send some data *)
    fbSend(Enable:=TRUE, Socket:=udiSocket, ValueAdr:=ADR(strSend), NoOfBytes:=LEN(strSend));
    IF fbSend.Done=TRUE THEN
        (* Send done *)
        fbSend(Enable:=FALSE);
        iState := 3;
    END_IF

3:
    (* Receive some data *)
    fbRecv(Enable:=TRUE, Socket:=udiSocket, ValueAdr:=ADR(strRecv), NoOfBytes:=SIZEOF(strRecv)-1);
    IF fbRecv.Done=TRUE THEN
        (* Received some data *)
        strRecv := LEFT(strRecv, UDINT_TO_INT(fbRecv.NoOfRecBytes)); (* end of string marker *)
        fbRecv(Enable:=FALSE);
        iState := 4;
    END_IF

4:
    (* Close the connection *)
    fbConnection(Enable:=FALSE);
    diSocket := SOCKET_INVALID;

END_CASE
```

TCP cyclic - example

The following example shows a simple TCO application cyclically exchanging a data structure between two devices using the RIL_SocketComm library.

Upon start, the program opens a TCP server on the port 50001 and waits for an incoming connection (server) or tries to establish a connection to the remote host (client). After the connection has been established, the data structure is cyclically exchanged between the two communication partners. The

program opens the server port again and waits for further incoming connections.

TCP server program

```
PROGRAM Client
VAR
    fbIL_TCPCyclic: IL_TCPCyclic;

    SendData: MY_STRUCTURE;
    RecvData: MY_STRUCTURE;

    bEnable: BOOL;
    bInOperation: BOOL;
    bError: BOOL;
    nErrorCode: ERROR_CODE;
    nErrorIdent: ERROR_STRUCT;
    nStatus: IL_CONN_STATUS;
    tResponseTime: TIME;
    bDoneSend: BOOL;
    bDoneRecv: BOOL;
END_VAR

fbIL_TCPCyclic(
    Enable:= bEnable,
    Mode:= Cyclic Client,
    IPAddress:= '192.168.0.10',
    Port:= 50001,
    SendDataAdr:= ADR(SendData),
    RecvDataAdr:= ADR(RecvData),
    SizeOfSendData:= SIZEOF(SendData),
    SizeOfRecvData:= SIZEOF(RecvData),
    Timeout:= T#1500MS,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> nErrorCode,
    ErrorIdent=> nErrorIdent,
    Status=> nStatus,
    ResponseTime=> tResponseTime,
    DoneSend=> bDoneSend,
    DoneRecv=> bDoneRecv);
```

TCP client program

```
PROGRAM Server
VAR
    fbIL_TCPCyclic: IL_TCPCyclic;

    SendData: MY_STRUCTURE;
    RecvData: MY_STRUCTURE;

    bEnable: BOOL;
    bInOperation: BOOL;
    bError: BOOL;
    nErrorCode: ERROR_CODE;
    nErrorIdent: ERROR_STRUCT;
    nStatus: IL_CONN_STATUS;
    tResponseTime: TIME;
    bDoneSend: BOOL;
    bDoneRecv: BOOL;
END_VAR

fbIL_TCPCyclic(
    Enable:= bEnable,
    Mode:= Cyclic Server,
    IPAddress:= '192.168.0.10',
    Port:= 50001,
    SendDataAdr:= ADR(SendData),
    RecvDataAdr:= ADR(RecvData),
    SizeOfSendData:= SIZEOF(SendData),
    SizeOfRecvData:= SIZEOF(RecvData),
    Timeout:= T#1500MS,
    InOperation=> bInOperation,
    Error=> bError,
    ErrorID=> nErrorCode,
    ErrorIdent=> nErrorIdent,
    Status=> nStatus,
    ResponseTime=> tResponseTime,
```

```
DoneSend=> bDoneSend,
DoneRecv=> bDoneRecv);
```

Data structure

```
// Example of a data structure
TYPE MY_STRUCTURE :
STRUCT
  bTestBool    : BOOL;
  byTestByte   : BYTE;
  uiTestUInt   : UINT;
  rTestReal    : REAL;
  tTestTime    : TIME;
  strTest      : STRING(22) := 'Hello World';
  dwTest       : DWORD;
END_STRUCT
END_TYPE
```

6.3.2 UDP

UDP is a connection-free IP protocol. In contrast to the TCP protocol, there is no connection establishment prior to the actual data exchange. Communication is package-based. Received data packages are stored in the buffer, but not merged. Every reception call takes exactly one package from the queue.

Received UDP data packages are not acknowledged. It is therefore not possible to identify lost packages. UDP is therefore suitable for cyclic data transmission, in which data of possibly lost packages is updated by later packages.

The following flow chart shows a typical mono-directional UDP client server configuration. The server opens a UDP socket at a given port while the UDP port of the client is unknown. In this configuration, the client can only send but not receive.

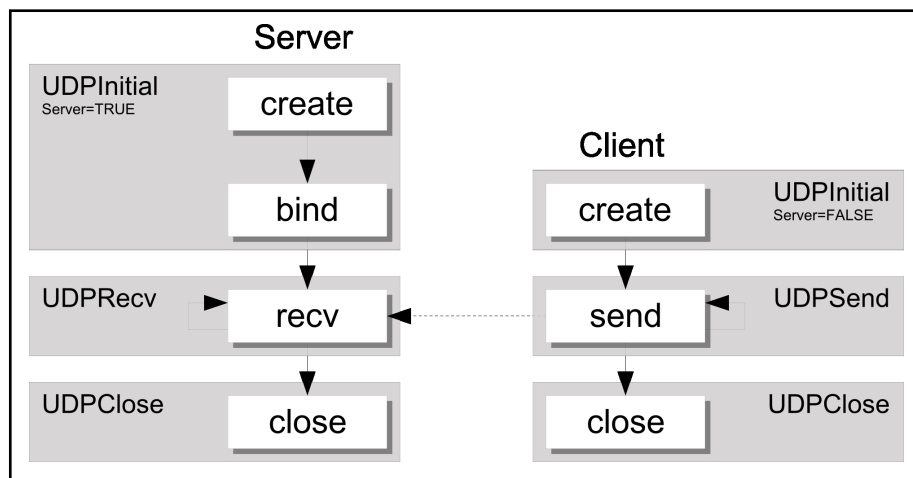


Fig. 6-3: UDP server-client flow chart

The following flow chart shows a typical bidirectional UDP server-server configuration. In this configuration, both communication partners can send and receive data packages, as the respective target port is known.

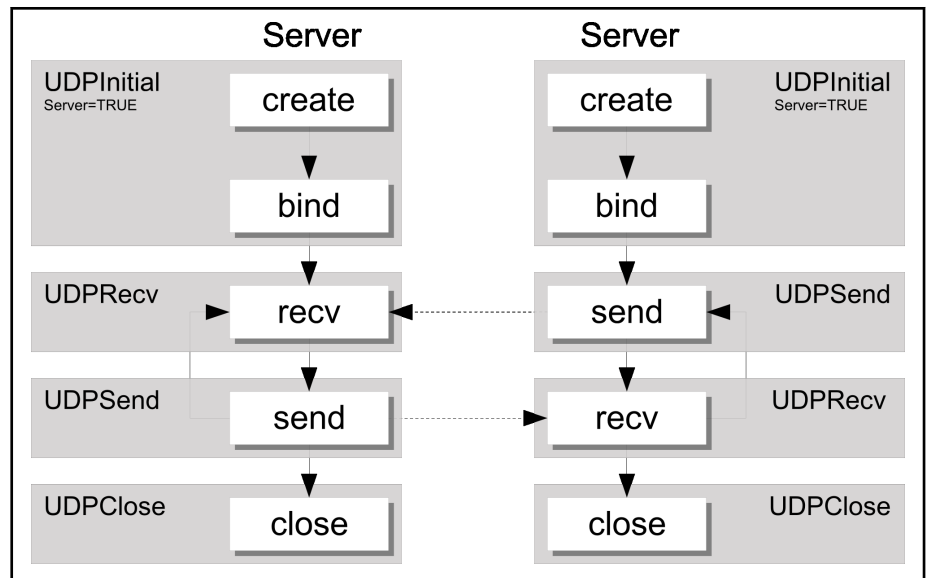


Fig. 6-4: UDP server-server flow chart

Asynchronous function blocks for a UDP connection

- [IL_UDPAsync on page 280](#)
- [IL_UDPIInitialAsync on page 282](#)
- [IL_UDPSendtoAsync on page 284](#)
- [IL_UDPRecvfromAsync on page 286](#)
- [IL_UDPCloseAsync on page 287](#)

Synchronous functions for a UDP connection

- [IL_UDPInitial on page 289](#)
- [IL_UDPSendto on page 290](#)
- [IL_UDPRecvfrom on page 291](#)
- [IL_UDPClose on page 292](#)

Details on implementation:

- [UDP asynchronous function blocks on page 279](#)
- [UDP synchronous functions on page 289](#)

For example programs to exchange data based on UDP, refer to:

- [UDP client example on page 251](#)
- [UDP server, synchronous transfer on page 252](#)
- [UDP client, synchronous transfer on page 254](#)

UDP client example

The following example shows a simple UDP client, which cyclically sends a data package once per second to a defined UDP server IP and port pair.

Type definition of a UDP package data structure:

```

TYPE UDP_PACKET :
STRUCT
    Sequence: UINT;
    State: INT;
    InfoTxt: STRING[20];
END_STRUCT
END_TYPE

```

Simple UDP client program (declaration)

```

(* This simple Example implements an UDP client. It sends an UDP packet once a second. *)
(* the packet can contain all process data and even strings. *)
(* A sequence number is implemented to allow for a missing data packets check *)

PROGRAM PLC_PRG
VAR CONSTANT
    (* communication settings *)
    strUDPServer: STRING := '192.168.1.254'; (* put in your remote UDP server IP here *)
    uiPort: UINT := 2323; (* remote UDP server port *)
END_VAR

VAR
    (* used function blocks *)
    fbUDPSocket: IL_UDPSocketAsync;
    fbUDPSend: IL_UDPSendtoAsync;

    (* variables *)
    udiSocket: UDINT; (* socket handle *)
    iCount: INT; (* send loop counter *)

    (* UDP packet data to send *)
    stUDPPacket: UDP_PACKET := (Sequence:=0, State:=1, InfoTxt:='Device Ready');
END_VAR

```

Simple UDP client program (implementation)

```

(* check for UDP socket *)
IF fbUDPSocket.Done = FALSE THEN
    (* no UDP socket yet => create one *)
    fbUDPSocket(Enable:=TRUE, Server:=FALSE, Socket=>udiSocket);

    IF fbUDPSocket.Error = TRUE THEN
        (* an error occurred => reset *)
        fbUDPSocket(Enable:=FALSE, Socket=>udiSocket);
    END_IF
ELSE
    IF iCount <= 0 THEN
        (* every 100 loops send UDP packet (10ms task * 100 loops = 1s) *)
        IF fbUDPSend.Active = FALSE THEN
            (* not already about to send => increase sequence number of UDP packet *)
            stUDPPacket.Sequence := stUDPPacket.Sequence + 1;
        END_IF

        (* send the packet *)
        fbUDPSend(Enable:=TRUE, Socket:=udiSocket,
            ValueAdr:=ADR(stUDPPacket), NoOfBytes:=SIZEOF(stUDPPacket),
            Address:=strUDPServer, Port:=uiPort);

        IF fbUDPSend.Done = TRUE THEN
            (* packet send => restart counter *)
            fbUDPSend(Enable:=FALSE);
            iCount := 100;
        ELSIF fbUDPSend.Error = TRUE THEN
            (* an error occurred => reset everything *)
            fbUDPSend(Enable:=FALSE);
            fbUDPSocket(Enable:=FALSE);
        END_IF
    ELSE
        (* decrease loop counter *)
        iCount := iCount - 1;
    END_IF
END_IF

```

UDP server, synchronous transfer

A UDP server is implemented in the following example. The UDP server receives data from a client and sends data to this client.

The connection is then closed and the socket is available.

The program requires the libraries "RIL_SocketComm" and "cmperrors".



- The function block IL_UDPInitial may not be called cyclically. The function block returns from the call, once the connection is established.
- IL_UDPInitial generates a socket upon each call. Consider the limitation of the number of sockets.
- If the error "ERR SOCK_WOULDBLOCK" occurs while sending or receiving data, the "Send" buffer is either full or the "Receive" buffer is still empty.

The error "ERR SOCK_WOULDBLOCK" can thus be ignored and the call repeated.

Program declaration

```
PROGRAM UDPSync_Server_Prog
VAR
  arRecvData: ARRAY[1..10] OF BYTE;
  arSendData: ARRAY[1..10] OF BYTE;

  uiState:      UINT := 0;
  i:            INT;
  udiResult:    UDINT;
  bResult:      BOOL;
  udiSocket:    UDINT;
  udiErrorNum:  UDINT;
  uiClient_Port:  UINT;
  strClient_Address: STRING;
  udiResult_Latched: UDINT;
  udiErrorNum_Latched: UDINT;
END_VAR

VAR CONSTANT
  STATE_INIT:      UINT := 0;
  STATE_SERVER:    UINT := 1;
  STATE_RECEIVE:   UINT := 2;
  STATE_SEND:      UINT := 3;
  STATE_CLOSE:     UINT := 4;
  STATE_ERROR:     UINT := 5;
  STATE_FINISH:    UINT := 6;
END_VAR
```

Implementation

```
//The following program implements an UDP Server
//the UDP Server receives data from a Client
//then the Server sends data back to the Client
//after that close connection and release socket

CASE uiState OF

  //Initialize senddata
  STATE_INIT:
    FOR i := 1 TO 10 BY 1 DO
      arSendData[i] := arSendData[i] + 1;
    END_FOR
    uiState := STATE_SERVER;

  //UDP Server
  STATE_SERVER:
    udiResult:= IL_UDPInitial(bServer:= TRUE, strAddress:= '', uiPort:= 6000, pudiSocket:= ADR(udiSocket),
      pudErrorNum:= ADR(udiErrorNum));

    //if no error, go to next state, else go to the error state
    IF udiResult = 0 AND udiErrorNum = 0 THEN
      uiState := STATE_RECEIVE;
    ELSE
      uiState := STATE_CLOSE;
    END_IF

  //Receive data from Client
  STATE_RECEIVE:
    udiResult:= IL_UDPRecvfrom(udiSocket:= udiSocket, pbyValueAdr:= ADR(arRecvData),
```

RIL_SocketComm.library

```

                pstSendSockAddr:= ADR(strClient_Address), puiSendPort:= ADR(uiClient_Port),
                pudtErrorNum:= ADR(udiErrorNum), udiNoOfBytes:= SIZEOF(arRecvData));

//go to the next state if the receive process is finished without error
IF udiResult <> 0 AND udiErrorNum = Errors.ERR_OK THEN
    uiState := STATE_SEND;
END_IF

//if error is 'wouldblock', try it again; else go to the error state
IF udiErrorNum <> 0 THEN
    IF udiErrorNum = Errors.ERR_SOCK_WOULDBLOCK THEN
        RETURN;
    ELSE
        uiState := STATE_CLOSE;
    END_IF
END_IF

//Send data to client
STATE_SEND:
    udiResult:= IL_UDPSendto(udiSocket:= udiSocket, pbyValueAdr:= ADR(arSendData),
        strAddress:= strClient_Address, uiPort:= uiClient_Port,
        pudtErrorNum:= ADR(udiErrorNum), udiNoOfBytes:= SIZEOF(arSendData));

//go to the next state if the send process is finished without error
IF udiResult <> 0 AND udiErrorNum = Errors.ERR_OK THEN
    uiState := STATE_CLOSE;
END_IF

//if error is wouldblock, try it again; else go to the error state
IF udiErrorNum <> 0 THEN
    IF udiErrorNum = Errors.ERR_SOCK_WOULDBLOCK THEN
        RETURN;
    ELSE
        uiState := STATE_CLOSE;
    END_IF
END_IF

//Release Socket
STATE_CLOSE:
    bResult := IL_UDPClose(udiSocket:= udiSocket, pudtErrorNum:= ADR(strClient_Address));

    //store error information
    IF udiErrorNum <> 0 THEN
        udiErrorNum_Latched := udiErrorNum;
    END_IF
    uiState := STATE_FINISH;

//store error information
STATE_ERROR:
    udiResult_Latched := udiResult;
    udiErrorNum_Latched := udiErrorNum;
    //go to the close state to break the connection and release the socket
    uiState := STATE_CLOSE;

//Finish, do nothing
STATE_FINISH:
    ;
END_CASE

```

UDP client, synchronous transfer

A UDP client is implemented in the following example. The UDP client sends data to a server and then receives data from this server.

The connection is then closed and the socket is available.

The program requires the libraries "RIL_SocketComm" and "cmpererrors".



- The function block IL_UDPInitial may not be called cyclically. The function block returns from the call, once the connection is established.
- IL_UDPInitial generates a socket upon each call. Consider the limitation of the number of sockets.
- If the error "ERR SOCK_WOULDBLOCK" occurs while sending or receiving data, the "Send" buffer is either full or the "Receive" buffer is still empty.

The error "ERR SOCK_WOULDBLOCK" can thus be ignored and the call repeated.

Program declaration

```
PROGRAM UDPSync_Client_Prog
VAR
  arRecvData: ARRAY[1..10] OF BYTE;
  arSendData: ARRAY[1..10] OF BYTE;

  uiState:      UINT := 0;
  i:            INT;
  udiResult:    UDINT;
  bResult:      BOOL;
  udiSocket:    UDINT;
  udiErrorNum:  UDINT;
  uiRecv_Port:  UINT;
  strRecv_Address: STRING;
  udiResult_Latched: UDINT;
  udiErrorNum_Latched: UDINT;
END_VAR

VAR CONSTANT
  STATE_INIT:      UINT := 0;
  STATE_CLIENT:    UINT := 1;
  STATE_RECEIVE:   UINT := 2;
  STATE_SEND:      UINT := 3;
  STATE_CLOSE:     UINT := 4;
  STATE_ERROR:     UINT := 5;
  STATE_FINISH:    UINT := 6;
END_VAR
```

Implementation

```
//The following program implements a UDP Client
//the UDP Client sends data to a Server
//then it receives data from the Server
//after that close the connection and release socket

CASE uiState OF

  //Initialize senddata
  STATE_INIT:
    FOR i := 1 TO 10 BY 1 DO
      arSendData[i] := arSendData[i] + 1;
    END_FOR
    uiState := STATE_CLIENT;

  //UDP Client
  STATE_CLIENT:
    udiResult := IL_UDPInitial(bServer:= FALSE, strAddress := '192.168.42.11', uiPort:= 6000,
      pudSocket:= ADR(udiSocket), pudErrorNum:= ADR(udiErrorNum));

    //if no error, go to next state, else go to the error state
    IF udiResult = 0 AND udiErrorNum = Errors.ERR_OK THEN
      uiState := STATE_SEND;
    ELSE
      uiState := STATE_ERROR;
    END_IF

  //Send data to Server
  STATE_SEND:
    udiResult:= IL_UDPSendto(udiSocket:= udiSocket, pbyValueAdr:= ADR(arSendData),
      strAddress:= '192.168.42.11', uiPort:= 6000,
      pudErrorNum:= ADR(udiErrorNum) , udiNoOfBytes:= SIZEOF(arSendData));
```

RIL_SocketComm.library

```

//go to the next state if the send process is finished without error
IF udiResult <> 0 AND udiErrorNum = Errors.ERR_OK THEN
    uiState := STATE_RECEIVE;
END_IF

//if error is wouldblock, try it again; else go to the error state
IF udiErrorNum <> 0 THEN
    IF udiErrorNum = Errors.ERR_SOCK_WOULDBLOCK THEN
        RETURN;
    ELSE
        uiState := STATE_ERROR;
    END_IF
END_IF

//Receive data from Server
STATE_RECEIVE:
    udiResult := IL_UDPRecvfrom(udiSocket := udiSocket, pbyValueAdr := ADR(arRecvData),
                                pstSendSockAddr := ADR(strRecv_Address), puiSendPort := ADR(uiRecv_Port),
                                pudErrorNum := ADR(udiErrorNum), udiNoOfBytes := SIZEOF(arRecvData));

//go to the next state if the receive process is finished without error
IF udiResult <> 0 AND udiErrorNum = Errors.ERR_OK THEN
    uiState := STATE_CLOSE;
END_IF

//if error is wouldblock, try it again; else go to the error state
IF udiErrorNum <> 0 THEN
    IF udiErrorNum = Errors.ERR_SOCK_WOULDBLOCK THEN
        RETURN;
    ELSE
        uiState := STATE_ERROR;
    END_IF
END_IF

//Release socket
STATE_CLOSE:
    bResult := IL_UDPClose(udiSocket := udiSocket, pudErrorNum := ADR(udiErrorNum));

//store error information
IF udiErrorNum <> 0 THEN
    udiErrorNum_Latched := udiErrorNum;
END_IF
uiState := STATE_FINISH;

//store error information
STATE_ERROR:
    udiResult_Latched := udiResult;
    udiErrorNum_Latched := udiErrorNum;
//go to the close state to break the connection and release the socket
uiState := STATE_CLOSE;

//Finish, do nothing
STATE_FINISH:
    ;
END_CASE

```

6.3.3 ICMP

The Internet Control Message Protocol (ICMP) exchanges information and error messages in networks via internet protocol (IP). It also provides a mechanism for simple testing of IP devices. This ping sends a short package to a remote host. The remote host responds with the same content. If the receiver receives the package from the remote host, the ping was successful.

Asynchronous function block for ICMP

- [IL_PingAsync on page 293](#),

Synchronous function for ICMP

- [IL_Ping on page 295](#),

6.3.4 FTP

The File Transfer Protocol (FTP) transfers files between two network devices based on the TCP protocol. An FTP client connects to an FTP server running

on the target computer. Once the connection is established, data can be read and written.

Upon firmware boot, an FTP server is started in all Rexroth controls and is therefore available as remote host. Windows-based computers normally do not provide an installed FTP server.

Usually, the FTP server runs on the TCP port 21. This port has to be reachable from the control. Any possible firewalls present in the network have to be configured accordingly.

Asynchronous function blocks for FTP:

- [IL_FTPGetAsync on page 296](#)
- [IL_FTPPutAsync on page 298](#)
- [IL_FTPGetFileAsync on page 301](#)
- [IL_FTPPutFileAsync on page 304](#)

Details on implementation: [Asynchronous FTP function blocks on page 295](#)

6.3.5 TFTP

Many peripheral devices use the TFTP protocol (Trivial File Transfer Protocol) to transfer firmware or the configuration. This protocol can be implemented using little resources and is thus used by devices with little memory size.

Upon firmware boot, an TFTP server is started in all Rexroth IndraDrives and is therefore available as remote host. This TFTP server is used for firmware updates. Windows-based computers normally do not provide an installed TFTP server.

Usually, the TFTP server runs on the UDP port 69. This port has to be reachable from the control. Any possible firewalls present in the network have to be configured accordingly.

Asynchronous function blocks for TFTP:

- [IL_TFTPGetFileAsync on page 306](#)
- [IL_TFTPPutFileAsync on page 309](#)

Details on implementation: [Asynchronous TFTP function blocks on page 306](#)

6.3.6 Cyclic communication

The function blocks under CyclicCommunication allow to exchange data cyclically. They are based on the TCP and UDP communication function blocks of the RIL_SocketComm.

Data to be exchanged can be specified in its user-defined data structures.

Apart from the cyclic communication, data can also be exchanged on request.

Asynchronous function blocks for the cyclic data exchange

- [IL_TCPCyclic on page 311](#)
- [IL_UDPCyclic on page 316](#)

Details on implementation: [Asynchronous communication function blocks for cyclic communication on page 311](#)

6.3.7 Byte order in IP-based communication

The byte order describes the memory organization for simple numerical values, primarily the archiving of integers. The smallest addressable unit is a byte. A number is stored in several bytes, whose memory addresses follow each other directly.

- With **big-endian** first byte order (literally "big end first"), the byte with the highest value bits (that are the most significant points) is stored first, i.e., at the smallest memory address.
- With **little-endian** first byte order (literally "small end first"), the byte with the lowest value bits (that are the least significant points) is stored at the smallest memory address.

The byte order for IP-based communication is based on the big-endian method. As controllers store data to the memory either in big-endian or in little-endian format, conversion functions adapted to the system are required. The functions are provided by the SysSocket library.

The following functions are available:

- SysSockHtons: Control to network 16 bits
- SysSockNtohs: Network to control 16 bits
- SysSockHtonl: Control to network 32 bits
- SysSockNtohl: Network to control 32 bits

The following data types have to be converted for transmission via TCP or UDP:

Data type	Control to network	Network to control
WORD, INT, UINT	SysSockHtons()	SysSockNtohs()
DWORD, DINT, UDINT, TIME	SysSockHtonl()	SysSockNtohl()

The data types BYTE, CHAR and STRING do not have to be converted.



The byte order for the data types REAL and LREAL is not specified. The correct setting can only be determined by trying.

6.3.8 Security - Disable unsecure protocols

See documentation "MLC 15VRS Functional Description", chapter "Protocols - System security".

6.4 TCP asynchronous function blocks

6.4.1 Introduction and overview

The TCP asynchronous function blocks are used in cyclic tasks and have to be called cyclically. Normally, multiple calls are required to complete an operation.

A function block starts with a rising edge at "Enable" and operates until either "Done" or "Error" is TRUE or "Enable" is reset. To start a new operation, reset "Enable" and set it again.

A new TCP connection can be established with IL_TCPConnectionAsync or IL_TCPInitialAsync.

IL_TCPConnectionAsync internally calls the function blocks IL_TCPInitialAsync and IL_TCPCloseAsync and maintains the socket handle of the connection.

A connection established with IL_TCPInitialAsync can be closed with IL_TCPCloseAsync.



IL_TCPCloseAsync may not be called for sockets established with IL_TCPConnectionAsync.

Once the connection has been established, data can be exchanged via the connection with IL_TCPSendAsync and IL_TCPRecvAsync.



TCP is a character stream-based protocol that guarantees the order and consistency of transmitted data. Received data is merged again internally to form a marking current. If a package-based communication is required, the user is responsible for identifying the package header.

Asynchronous function blocks for a TCP connection:

- [IL_TCPConnectionAsync on page 259](#)
- [IL_TCPInitialAsync on page 263](#)
- [IL_TCPSendAsync on page 267](#)
- [IL_TCPRecvAsync on page 268](#)
- [IL_TCPRecvPacketAsync on page 270](#)
- [IL_TCPCloseAsync on page 272](#)

6.4.2 IL_TCPConnectionAsync

Brief description The function block IL_TCPConnectionAsync establishes either a TCP server or a TCP client depending on the "server" input.

Interface description

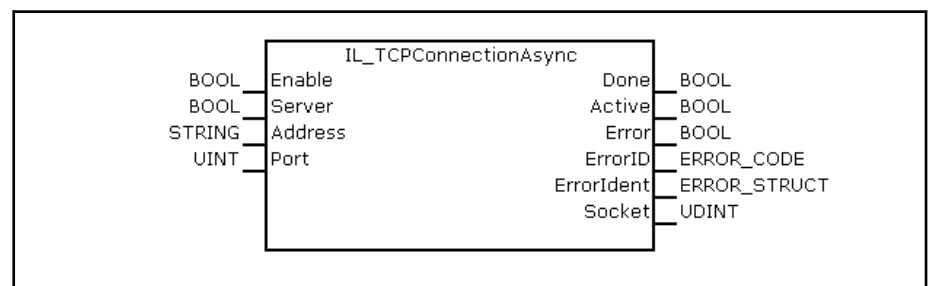


Fig. 6-5: IL_TCPConnectionAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Server	BOOL	TRUE: TCP server is created. FALSE: TCP client is created.
	Address	STRING	If "Server" is set, this input is not used and the own IP address is used. If "Server" is FALSE, this input contains the IP address of the device to be connected.
	Port	UINT	The TCP port at which incoming connections are expected or to which they are to be connected.

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	TRUE indicates that the connection was successfully established.
	Active	BOOL	TRUE indicates that the function block is waiting for the connection to be established.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 262)
	Socket	UDINT	Contains the socket handle of the new connection.

Tab. 6-1: IL_TCPConnectionAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Server	BOOL			FALSE	Rising edge at "Enable"
Address	STRING			"	Rising edge at "Enable"
Port	UINT	0	65535	0	Rising edge at "Enable"

Tab. 6-2: Min./max. and default values of the IL_TCPConnectionAsync function block inputs

Functional description

The function block IL_TCPConnectionAsync establishes either a TCP server or a TCP client depending on the "server" input. Processing starts at a rising edge at "Enable". The function block has to be called cyclically until "Done" or "Error" is TRUE.

In server mode, the function block opens a socket at the TCP port specified in "Port". If another TCP client connects to this port, the connection is established. The socket handle belonging to this connection is displayed at the "Socket" output and the output "Done" is set. While the system is waiting for an incoming connection, the "Active" output is set.

In client mode, the function block connects to the IP address given in "Address" on the TCP target port given in "Port". After a successfully established connection, the socket handle is displayed at the "Socket" output and the "Done" output is set. While a connection is established, the "Active" output is set.

Once a connection is established, the "Socket" output contains the socket handle of the connection.

In case of error, the "Error" output is set. SOCKET_INVALID is displayed at the "Socket" output. The error class and the exact error code can be retrieved at the "ErrorID" and "ErrorIdent" outputs.

If "Enable" is set to FALSE before "Done" is TRUE, no connection is established and all internal resources are enabled. If "Enable" is set to FALSE after a connection was established, the connection is closed. Possible errors during the connection abort are handled internally and are not issued at the error outputs. After "Enable" was set to FALSE, the "Socket" output is set to SOCKET_INVALID.

The "Active" output can be used as a timeout indicator during connection establishment. Typical timeouts for connection establishment range from 2 to 10 seconds.

IL_TCPCloseAsync is not to be used on sockets created with IL_TCPCon-
nectionAsync.

Signal-time diagrams

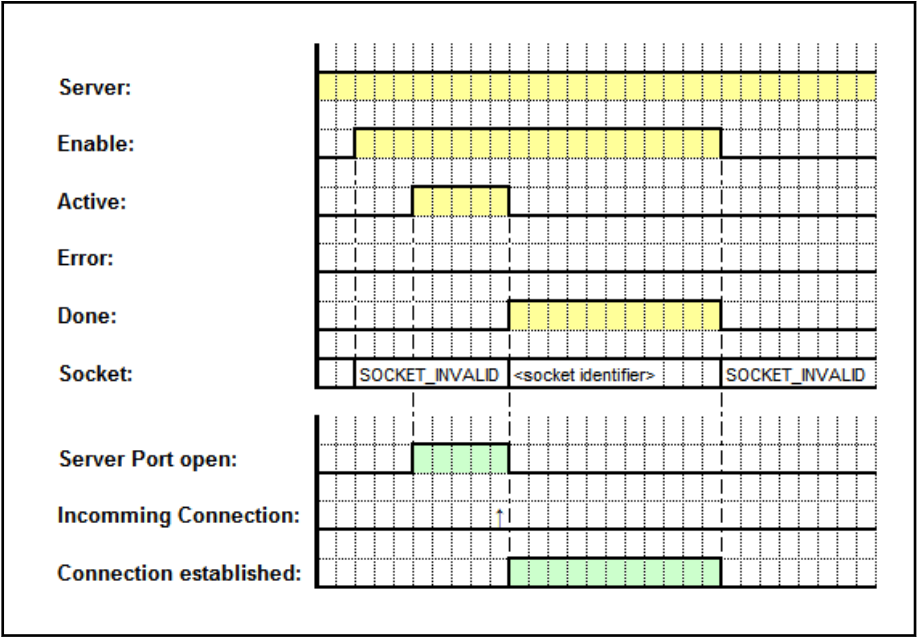


Fig. 6-6: Time diagram of a TCP server with an incoming connection

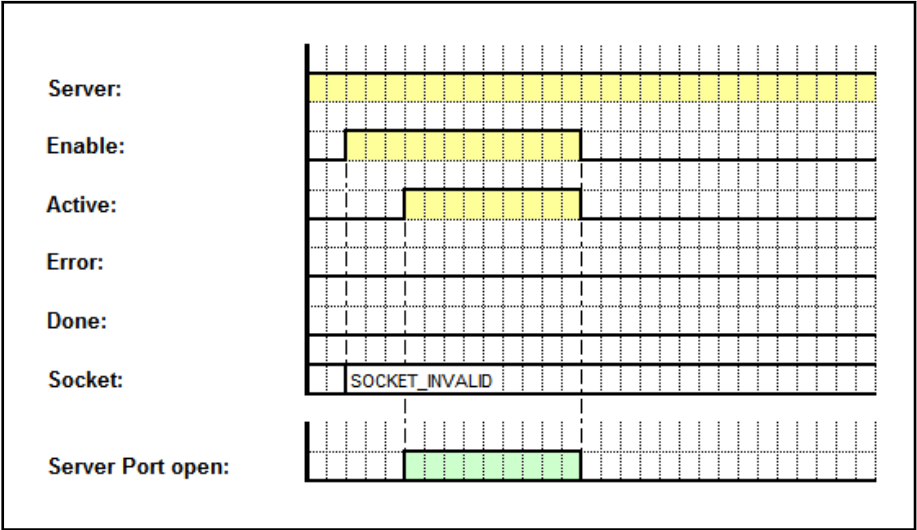


Fig. 6-7: Time diagram of a TCP server without incoming connection

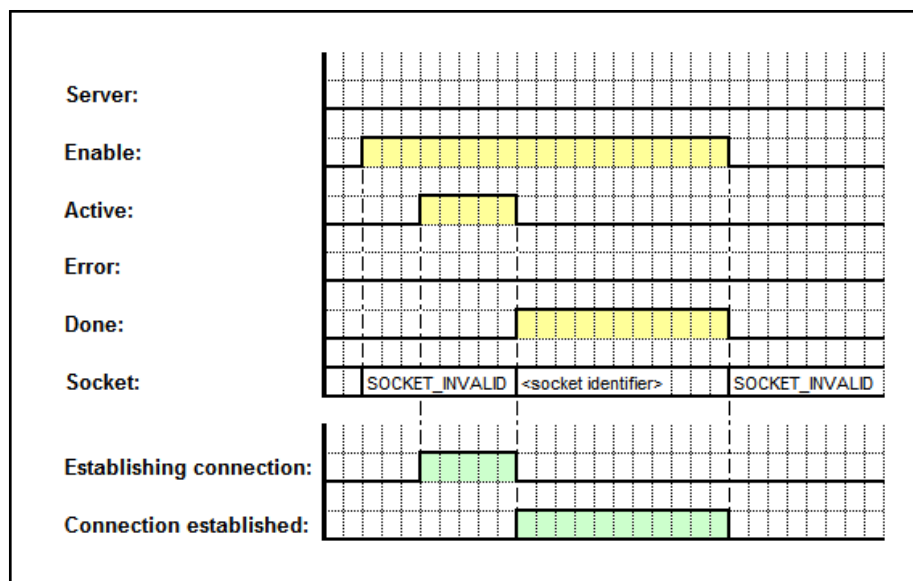


Fig. 6-8: Time diagram of a TCP client, connected to another server

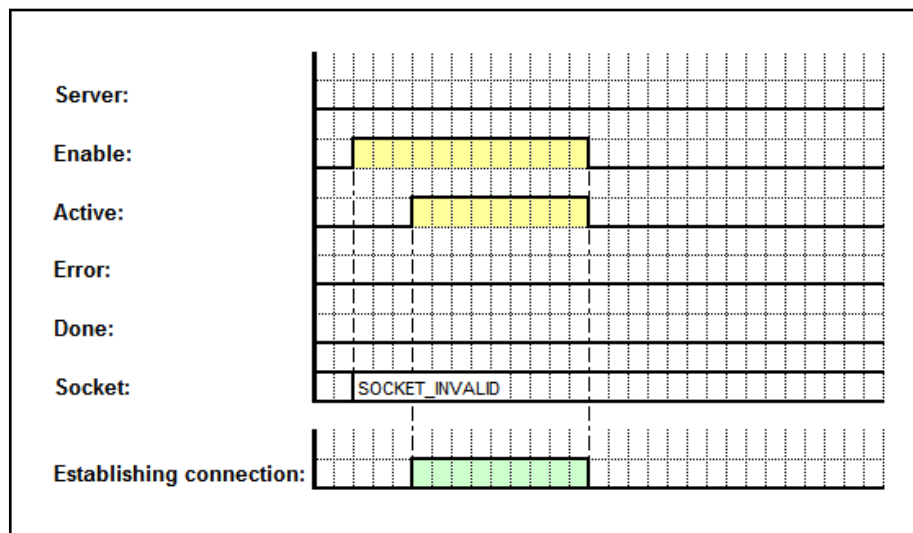


Fig. 6-9: Time diagram of a TCP client, which cannot connect to another server

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not go into list mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	16#00000000	No route to target device

Tab. 6-3: Error codes generated by IL_TCPConnectionAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.4.3 IL_TCPInitialAsync

Brief description The function block IL_TCPInitialAsync establishes either a TCP server or a TCP client depending on the "server" input.

Interface description

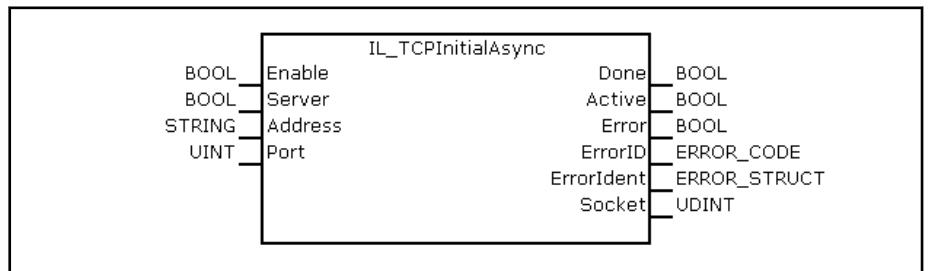


Fig. 6-10: IL_TCPInitialAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Server	BOOL	TRUE: TCP server is created FALSE: TCP client is created
	Address	STRING	If "Server" is set, this input is not used and the own IP address is used. If "Server" is FALSE, this input contains the IP address of the device to be connected.
	Port	UINT	The TCP port at which incoming connections are expected or to which they are to be connected.
VAR_OUTPUT	Done	BOOL	TRUE indicates that the connection was successfully established.
	Active	BOOL	TRUE indicates that the function block is waiting for the connection to be established.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 266)
	Socket	UDINT	Contains the socket handle of the new connection.

Tab. 6-4: IL_TCPInitialAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Server	BOOL			FALSE	Rising edge at "Enable"

Name	Type	Min. value	Max. value	Default value	Effective
Address	STRING			"	Rising edge at "Enable"
Port	UINT	0	65535	0	Rising edge at "Enable"

Tab. 6-5: Min./max. and default values of the IL_TCPInitialAsync function block inputs

Functional description

The function block IL_TCPInitialAsync establishes either a TCP server or a TCP client depending on the "server" input. Processing starts at a rising edge at "Enable". The function block has to be called cyclically until "Done" or "Error" is TRUE.

In server mode, the function block opens a socket at the TCP port specified in "Port". If another TCP client connects to this port, the connection is established. The socket handle belonging to this connection is displayed at the "Socket" output and the output "Done" is set. While the system is waiting for an incoming connection, the "Active" output is set.

In client mode, the function block connects to the IP address specified in "Address" on the TCP target port specified in "Port". After a successfully established connection, the socket handle is displayed at the "Socket" output and the "Done" output is set. While a connection is established, the "Active" output is set.

Once a connection is established, the "Socket" output contains the socket handle of the connection.

In case of error, the "Error" output is set. SOCKET_INVALID is displayed at the "Socket" output. The error class and the exact error code can be retrieved at the "ErrorID" and "ErrorIdent" outputs.

If "Enable" is set to FALSE before "Done" is TRUE, no connection is established and all internal resources are enabled. After "Enable" was set to FALSE, the "Socket" output is set to SOCKET_INVALID.

The "Active" output can be used as a timeout indicator during connection establishment. Typical timeouts for connection establishment range from 2 to 10 seconds.

The IL_TCPCloseAsync function block has to be used to close connections created with IL_TCPInitialAsync.

Signal-time diagrams

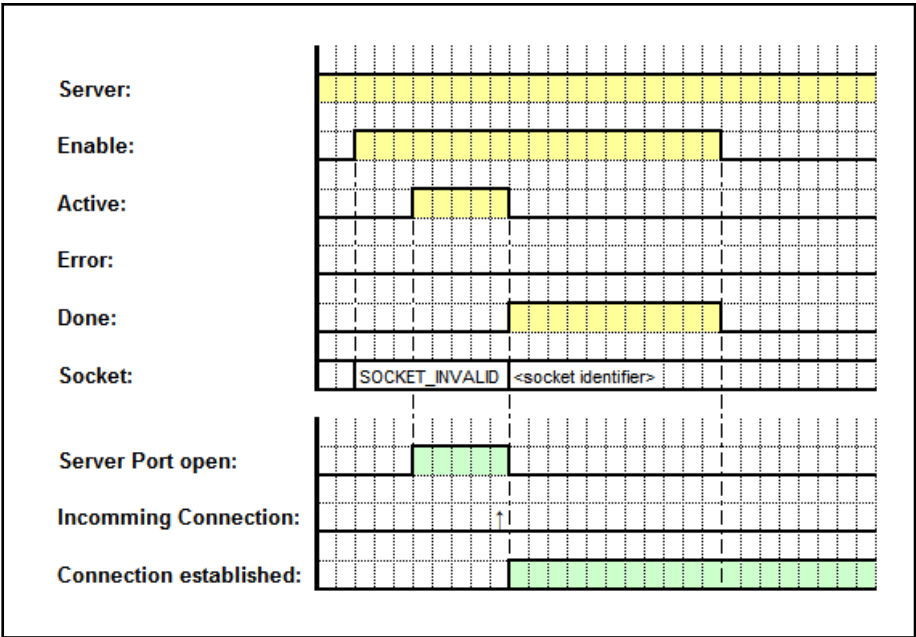


Fig. 6-11: Time diagram of a TCP server with an incoming connection

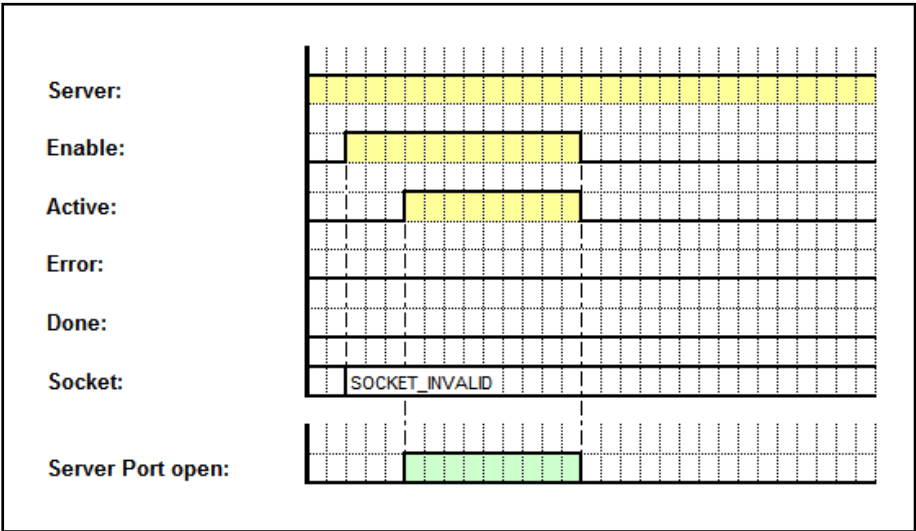


Fig. 6-12: Time diagram of a TCP server without incoming connection

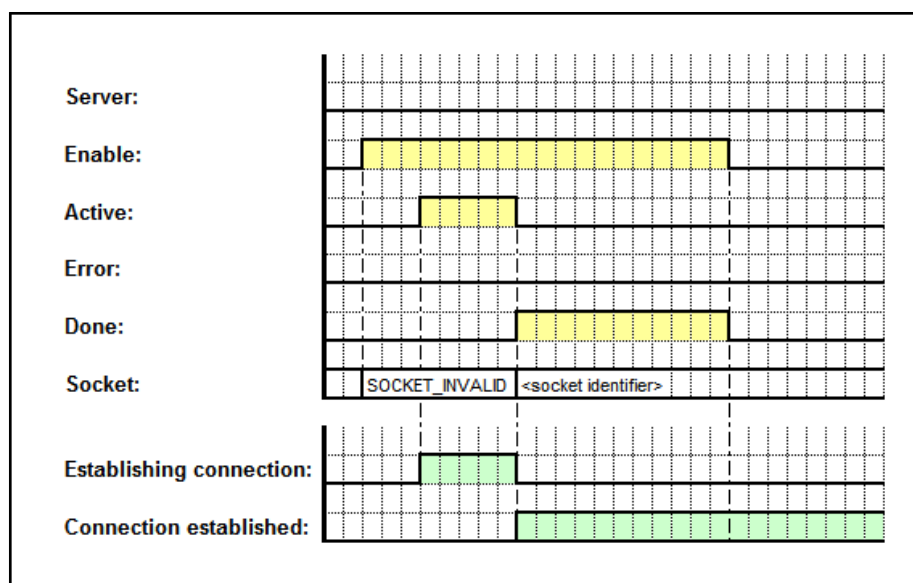


Fig. 6-13: Time diagram of a TCP client, connected to another server

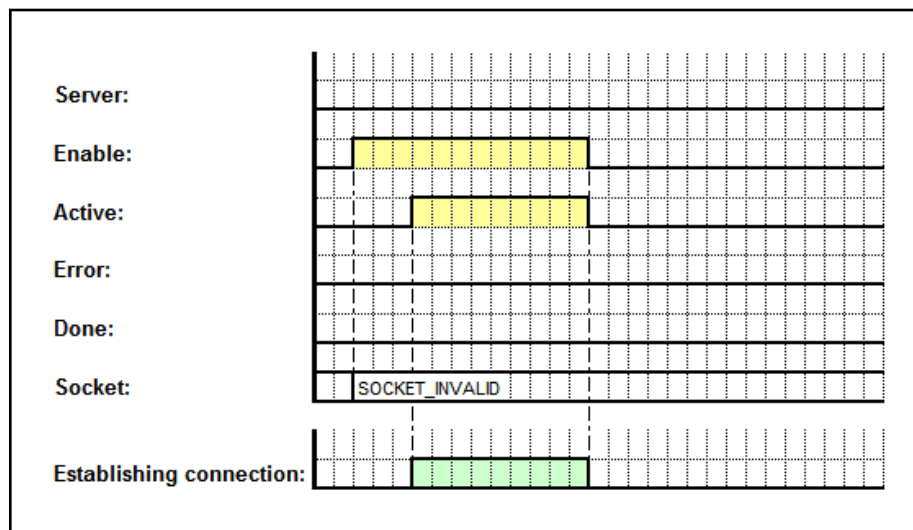


Fig. 6-14: Time diagram of a TCP client, which cannot connect to another server

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not go into list mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	16#00000000	No route to target device

Tab. 6-6: Error codes generated by IL_TCPInitialAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.4.4 IL_TCPSendAsync

Brief description The function block IL_TCPSendAsync sends data via a connection established with IL_TCPConnectionAsync or IL_TCPInitialAsync.

Interface description

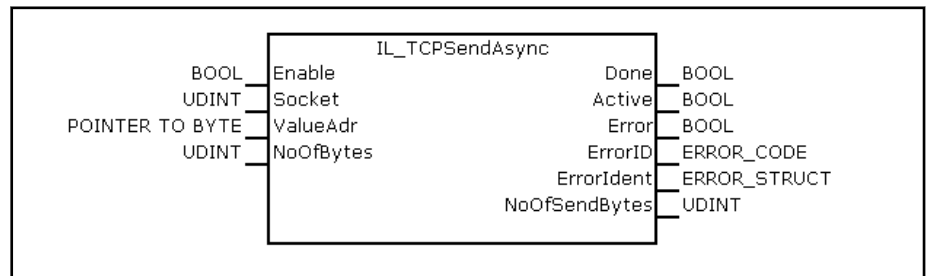


Fig. 6-15: IL_TCPSendAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge Binary outputs are reset at a falling edge.
	Socket	UDINT	IL_TCPConnectionAsync or IL_TCPInitialAsync socket handle
	ValueAdr	POINTER TO BYTE	Initial address of data to be sent
	NoOfBytes	UDINT	Number of characters to be sent
VAR_OUTPUT	Done	BOOL	TRUE indicates that all data was successfully sent.
	Active	BOOL	TRUE indicates that the function block is sending data.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 268)
	NoOfSend-Bytes	UDINT	Number of bytes actually sent

Tab. 6-7: IL_TCPSendAsync Interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"

Name	Type	Min. value	Max. value	Default value	Effective
ValueAdr	POINTER TO BYTE			0	Rising edge at "Enable"
NoOfBytes	UDINT	0	16#FFFFFFFF	0	Rising edge at "Enable"

Tab. 6-8: Min./max. and default values of the IL_TCPSendAsync function block inputs

Functional description The function block IL_TCPSendAsync sends data via a connection created with IL_TCPConnectionAsync or IL_TCPInitialAsync. Transmission begins at a rising edge at "Enable".



The function block has to be called cyclically until "Done" or "Error" is TRUE.

The buffer to be sent is applied to the "ValueAdr" input. The number of characters to be sent is applied at the "NoOfBytes" input.

If "Enable" is reset before "Done" is TRUE, the transmission is interrupted. The number of bytes already sent can be retrieved at the "NoOfSentBytes" output.

The "Active" output is set during data transmission. "Active" can be used as a timeout indicator. Typical transmission timeouts range from 1 to 2 seconds. During transmission ("Active" is TRUE), the number of bytes already sent can be retrieved at the "NoOfSentBytes" output. Generally all bytes are sent in one call.



IL_TCPSendAsync ensures that all bytes were sent before setting "Done".

IL_TCPSendAsync detects an interrupted connection. This is displayed by the relevant error code.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid
COMMUNICATION_ERROR	16#0000020E	16#00000000	Connection was closed
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection

Tab. 6-9: Error codes generated by IL_TCPSendAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.4.5 IL_TCPRecvAsync

Brief description The function block IL_TCPRecvAsync receives data via a connection established with IL_TCPConnectionAsync or IL_TCPInitialAsync.

Interface description

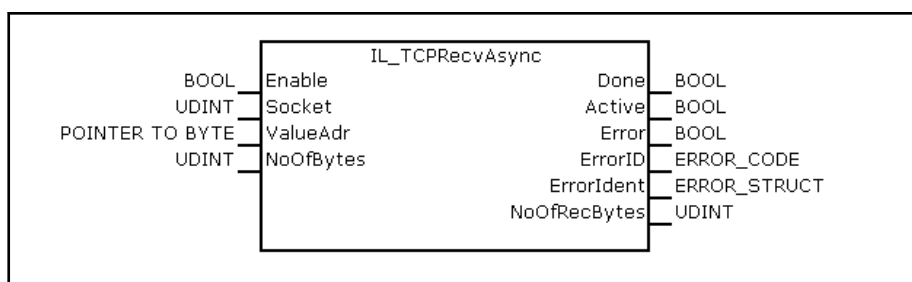


Fig. 6-16: IL_TCPRecvAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Socket	UDINT	IL_TCPConnectionAsync or IL_TCPIInitialAsync socket handle
	ValueAdr	POINTER TO BYTE	Initial address of the reception buffer
	NoOfBytes	UDINT	Size of the received data buffer
VAR_OUTPUT	Done	BOOL	TRUE indicates that all data was successfully received.
	Active	BOOL	TRUE indicates that the function block is waiting for incoming data.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 270)
	NoOfRecBytes	UDINT	Number of bytes received

Tab. 6-10: IL_TCPRecvAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Enable"
NoOfBytes	UDINT	0	16#FFFFFFFF	0	Rising edge at "Enable"

Tab. 6-11: Min./max. and default values of the IL_TCPRecvAsync function block inputs

Functional description

The function block IL_TCPRecvAsync receives data via a connection established with IL_TCPConnectionAsync or IL_TCPIInitialAsync.

Transmission begins at a rising edge at "Enable".



The function block has to be called cyclically until "Done" or "Error" is TRUE.

If "Enable" is reset before "Done" is TRUE, the transmission is interrupted. In case of such an abort, no data is accepted. "NoOfRecBytes" is hence 0.

The "Active" output is set during data transmission.

Active remains TRUE until data is received.

The number of bytes received can be lower than the buffer size.

If the exact buffer size is to be received, the function block [IL_TCPRecvPacketAsync](#) on page 270 has to be used. "Active" can be used as a timeout indicator.

If the number of the bytes sent by the remote host is higher than the buffer size, only the number of bytes entered into "NoOfBytes" are copied to the buffer. The remaining bytes are copied at next call.



"Active" can be used as a timeout indicator. Typical reception timeouts range from 1 to 2 seconds.

IL_TCPRecvAsync detects a connection abort. This is displayed by the relevant error code.

Signal-time diagram

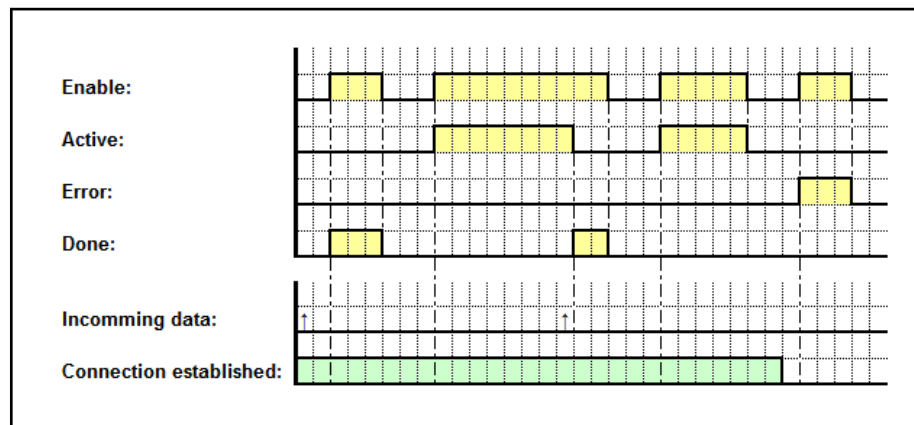


Fig. 6-17: Data reception time diagram

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid.
COMMUNICATION_ERROR	16#0000020E	16#00000000	Connection was closed.
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection.

Tab. 6-12: Error codes generated by IL_TCPRecvAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.4.6 IL_TCPRecvPacketAsync

Brief description

The function block IL_TCPRecvPacketAsync receives a defined number of bytes via a connection established with IL_TCPConnectionAsync or IL_TCPI-nitialAsync.

Interface description

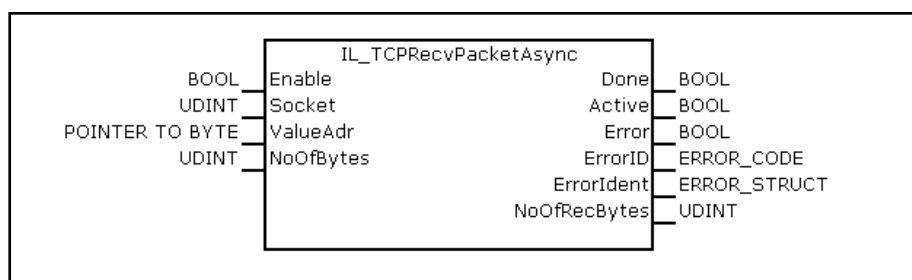


Fig. 6-18: IL_TCPRecvPacketAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge Binary outputs are reset at a falling edge.
	Socket	UDINT	IL_TCPConnectionAsync or IL_TCPIInitialAsync socket handle
	ValueAdr	POINTER TO BYTE	Initial address of the reception buffer
	NoOfBytes	UDINT	Number of bytes to be received
VAR_OUTPUT	Done	BOOL	TRUE indicates that all data was successfully received.
	Active	BOOL	TRUE indicates that the function block is waiting for incoming data.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 272)
	NoOfRecBytes	UDINT	Number of bytes copied to the reception buffer

Tab. 6-13: IL_TCPRecvPacketAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Enable"
NoOfBytes	UDINT	0	16#FFFFFFFF	0	Rising edge at "Enable"

Tab. 6-14: Min., max. and default values of the inputs of the IL_TCPRecvPacketAsync function block

Functional description

The function block IL_TCPRecvPacketAsync receives a **defined number** of bytes via a connection established with IL_TCPConnectionAsync or IL_TCPIInitialAsync.

Transmission begins at a rising edge at "Enable".



The function block has to be called cyclically until "Done" or "Error" is TRUE.

If "Enable" is reset before "Done" is TRUE, the transmission is canceled. In case of such an abort, no data is accepted. "NoOfRecBytes" is hence 0.

The "Active" output is set during data transmission.

1. If the amount of data is smaller than the buffer size, it is completely transferred. For the actual amount, refer to "NoOfRecBytes". ACTIVE remains TRUE until NoOfRecBytes < NoOfBytes.
2. If the amount of data is equal to the buffer size, it is transferred and NoOfRecBytes = NoOfBytes. ACTIVE is set to FALSE and DONE to TRUE.
3. If the amount of data is higher than the buffer size, only NoOfRecBytes is transferred. ACTIVE is set to FALSE and DONE to TRUE.

Upon next function block call, the remaining bytes are transferred and might be followed by data of the next package to be transferred.

Note! There can be a buffer overflow on the complete transmission path!

IL_TCPRecvPacketAsync detects an interrupted connection. This is displayed by the relevant error code.

Signal-time diagram

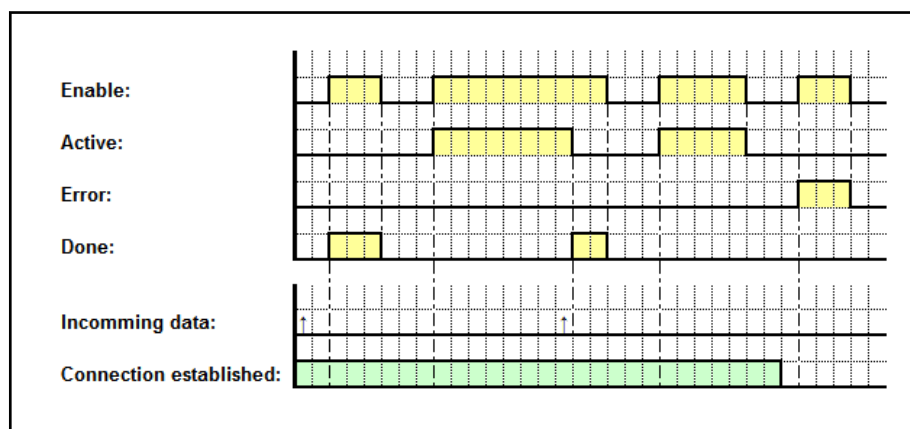


Fig. 6-19: Data reception time diagram

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid.
COMMUNICATION_ERROR	16#0000020E	16#00000000	Connection was closed.
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection.

Tab. 6-15: Error codes generated by IL_TCPRecvPacketAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.4.7 IL_TCPCloseAsync

Brief description The function block IL_TCPCloseAsync closes a connection created with IL_TCPInitialAsync.

Interface description

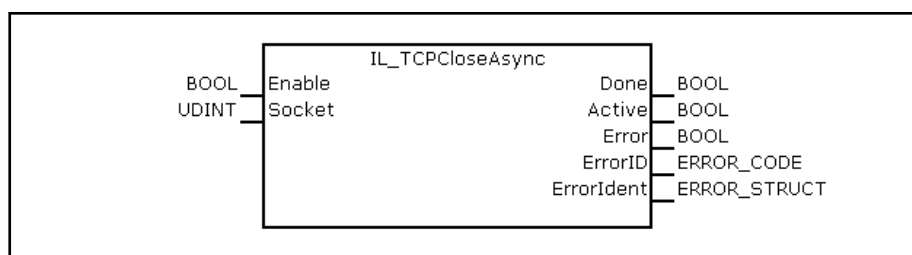


Fig. 6-20: IL_TCPCloseAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Socket	UDINT	IL_TCPInitialAsync socket handle
VAR_OUTPUT	Done	BOOL	TRUE indicates that the socket was successfully closed.
	Active	BOOL	TRUE indicates that the socket is closed.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 273)

Tab. 6-16: IL_TCPCloseAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"

Tab. 6-17: Min./max. and default values of the IL_TCPCloseAsync function block inputs

Functional description

The function block IL_TCPCloseAsync closes a connection created with IL_TCPInitialAsync.

The function block could require more calls than one to terminate the operation. The operation is completed successfully, if the "Done" output is set. "Active" is set while the operation is running. If an error occurs, the "Error" output is set. The outputs "ErrorID" and "ErrorIdent" contain more detailed error information.



The function block has to be called after a rising edge at "Enable" until either "Done" or "Error" are set.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection

Tab. 6-18: Error codes generated by IL_TCPCloseAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.5 TCP synchronous functions

6.5.1 Introduction and overview

The synchronous TCP functions are implemented as blocking calls. Thus, they do not return until the operation is completed or an error occurred. These functions are only to be used in free running tasks.

IL_TCPInitial establishes a connection and returns a socket handle. If the socket handle is valid, the connection was established. After the connection has been establishment, data can be sent and received via IL_TCPSend and IL_TCPRecv.

IL_TCPClose closes an existing connection and enables the socket handle.



TCP is a character stream-based protocol that guarantees the order and consistency of transmitted data. Received data is merged again internally to form a marking current. If a package-based communication is required, the user is responsible for identifying the package header.

Synchronous functions for a TCP connection:

- [IL_TCPInitial on page 274](#)
- [IL_TCPSend on page 276](#)
- [IL_TCPRecv on page 277](#)
- [IL_TCPRecvPacket on page 278](#)
- [IL_TCPClose on page 279](#)

6.5.2 IL_TCPInitial

Brief description

The IL_TCPInitial function establishes either a TCP server or a TCP client depending on the "bServer" parameter.

Interface description

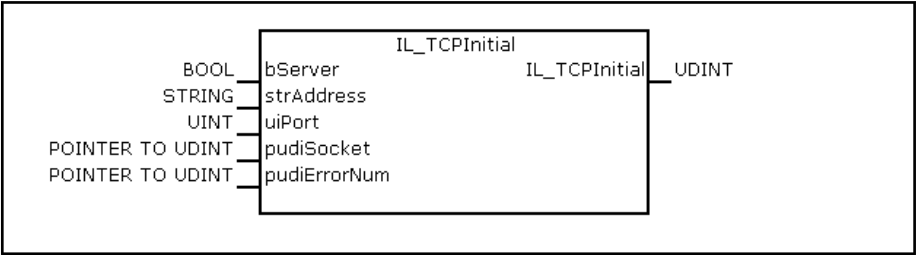


Fig. 6-21: IL_TCPInitial function

	Name	Type	Description
VAR_INPUT	bServer	BOOL	TRUE: TCP server is created FALSE: TCP client is created
	strAddress	STRING	If "bServer" is set, this input is not used and the own IP address is used. If "bServer" is FALSE, this input contains the IP address of the device to be connected.
	uiPort	UINT	The TCP port at which incoming connections are expected or to which they are to be connected.
	pdiSocket	POINTER TO UDINT	Contains the socket handle of the new connection.
	pdiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 275).
RETURN		UDINT	0 if a connection was successfully established; otherwise INVALID_SOCKET.

Tab. 6-19: IL_TCPInitial interface

Functional description

The IL_TCPInitial function establishes either a TCP server or a TCP client depending on the "bServer" parameter.

In server mode, the function opens a socket at the TCP port given in "uiPort". If another TCP client connects to this port, the connection is established. The socket handle belonging to the connection is stored in "pdiSocket" and the function returns with the return value 0.

In client mode, the function connects to the IP address provided in "strAddress" at the TCP target port provided in "uiPort". If a connection was established successfully, the socket handle is archived in "pdiSocket" and the function returns with the return value 0. If no connection can be established within 2 seconds, the function returns with the return value SOCKET_INVALID and the "pdiErrorNum" contains a corresponding error code.

If an error occurs, the function returns with SOCKET_INVALID and stores the error code in "pdiErrorNum".

Troubleshooting

The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pdiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	-	Could not create socket
OTHER_ERROR	16#00000041	-	Error when setting I/O mode
OTHER_ERROR	16#00000042	-	Could not connect socket to port
OTHER_ERROR	16#00000043	-	Socket could not go into list mode.
OTHER_ERROR	16#00000044	-	Could not accept incoming connection.
OTHER_ERROR	16#00000045	-	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	-	Target address invalid
COMMUNICATION_ERROR	16#00000209	-	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	-	No route to target device

Tab. 6-20: IL_TCPInitial error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150 on page 322](#).

6.5.3 IL_TCPSend

Brief description The IL_TCPSend function sends data via a connection created with IL_TCPI-initial.

This function does not return until either all data was transmitted or an error occurs.

Interface description

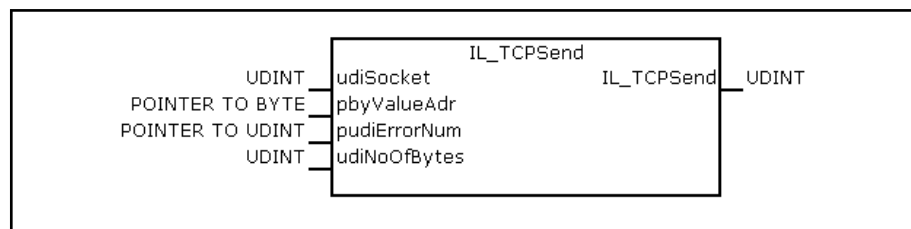


Fig. 6-22: IL_TCPSend function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_TCPIinitial socket handle
	pbyValueAdr	POINTER TO BYTE	Initial address of data to be sent.
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 276).
	udiNoOfBytes	UDINT	Number of characters to be sent
RETURN		UDINT	Number of bytes actually sent 0 indicates that the connection was closed or an error occurred.

Tab. 6-21: IL_TCPSend interface

Functional description The IL_TCPSend function sends data via a connection created with IL_TCPI-initial. This function does not return until either all data was transmitted or an error occurred.

The buffer to be sent is created in "pbyValueAdr". The number of characters to be sent is transferred in "udiNoOfBytes".

After successful transmission, this function returns the number of characters transmitted. If 0 is returned, the connection was closed before data could be sent. In case of error, the error code is stored to "pudiErrorNum" and the function returns with a return value of 0.

Troubleshooting The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	-	"ValueAdr" input is 0
INPUT_RANGE_ERROR	16#00000005	-	"NoOfBytes" input is 0
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid.
COMMUNICATION_ERROR	16#0000020E	-	Connection was closed.
COMMUNICATION_ERROR	16#00000211	-	Remote host closed the connection.

Tab. 6-22: IL_TCPSend error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150 on page 322](#).

6.5.4 IL_TCPRecv

Brief description The IL_TCPRecv function receives data via a connection created with IL_TCPInitial. This function does not return until either data was received or an error occurred.

Interface description

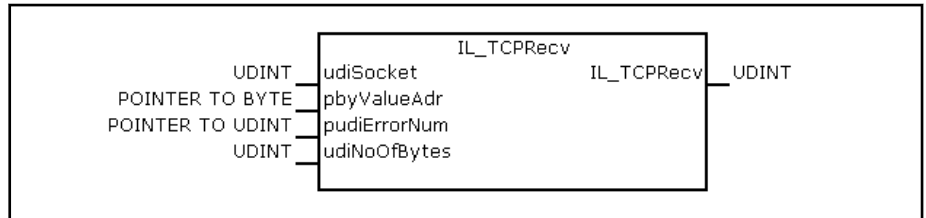


Fig. 6-23: IL_TCPRecv function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_TCPInitial socket handle
	pbyValueAdr	POINTER TO BYTE	Initial address of the received data buffer
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 277).
	udiNoOfBytes	UDINT	Size of the received data buffer
RETURN		UDINT	Number of bytes received. 0 indicates that the connection was closed or an error occurred.

Tab. 6-23: IL_TCPRecv interface

Functional description The IL_TCPRecv function receives data via a connection created with IL_TCPInitial. This function does not return until either data was received or an error occurred.

The reception buffer is created in "pbyValueAdr". The size of the reception buffer is transmitted in "udiNoOfBytes".

The IL_TCPRecv function receives characters up to the maximum size of the reception buffer connected. Fewer characters can also be received. If the complete buffer is always to be filled, the function [IL_TCPRecvPacket on page 278](#) is to be used. After successful transmission, this function returns the number of characters received. If 0 is returned, the connection was closed before data could be received. In case of error, the error code is stored to "pudiErrorNum" and the function returns with a return value of 0.

If no data is received within a second, the function returns with the return value 0 and stores the error code 16#0000020A in "pudiErrorNum".

Troubleshooting The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	-	"ValueAdr" input is 0
INPUT_RANGE_ERROR	16#00000005	-	"NoOfBytes" input is 0
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid.
COMMUNICATION_ERROR	16#0000020A	-	No data received within the timeout
COMMUNICATION_ERROR	16#0000020E	-	Connection was closed.
COMMUNICATION_ERROR	16#00000211	-	Remote host closed the connection.

Tab. 6-24: IL_TCPRecv error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.5.5 IL_TCPRecvPacket

Brief description The IL_TCPRecvPacket function receives data via a connection created with IL_TCPInitial. This function does not return until either data was received or an error occurred.

Interface description

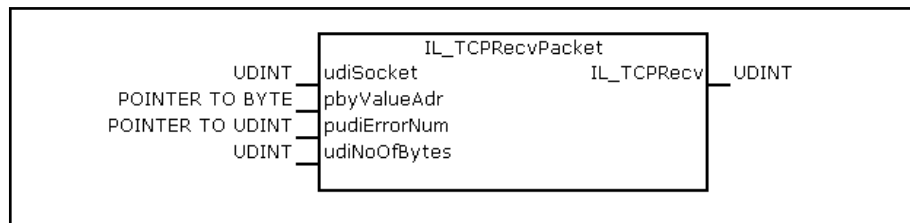


Fig. 6-24: IL_TCPRecvPacket function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_TCPInitial socket handle
	pbyValueAdr	POINTER TO BYTE	Initial address of the received data buffer
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 278).
	udiNoOfBytes	UDINT	Size of the received data buffer
RETURN VALUE		UDINT	Number of bytes received. 0 indicates that either the connection was closed or an error occurred.

Tab. 6-25: IL_TCPRecvPacket interface

Functional description The IL_TCPRecvPacket function receives data via a connection created with IL_TCPInitial. This function does not return until either data was received or an error occurred.

The reception buffer is created in "pbyValueAdr". The size of the reception buffer is transmitted in "udiNoOfBytes".

The IL_TCPRecvPacket function receives characters until the connected reception buffer is full.

It returns when the reception buffer is full or a timeout occurred. After successful transmission, this function returns the number of characters received.

If 0 is returned, the connection was closed before data could be received. In case of error, the error code is stored to "pudiErrorNum" and the function returns with a return value of 0.

If no data is received within a second, the function returns with the return value 0 and stores the error code 16#0000020A in "pudiErrorNum".

Troubleshooting The function uses the error table [ETHERNET_TABLE](#), 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	-	"ValueAdr" input is 0
INPUT_RANGE_ERROR	16#00000005	-	"NoOfBytes" input is 0
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid.
COMMUNICATION_ERROR	16#0000020A	-	No data received within the timeout

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#0000020E	-	Connection was closed.
COMMUNICATION_ERROR	16#00000211	-	Remote host closed the connection.

Tab. 6-26: IL_TCPRecvPacket error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.5.6 IL_TCPClose

Brief description The function IL_TCPClose closes a connection created with IL_TCPInitial. This function returns immediately.

Interface description

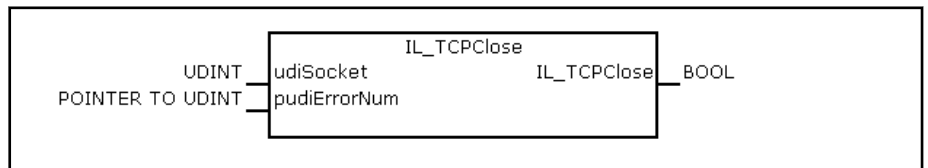


Fig. 6-25: IL_TCPClose function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_TCPInitial socket handle
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 279).
RETURN		BOOL	TRUE: Socket was closed successfully. FALSE: Error occurred when the socket was closed.

Tab. 6-27: IL_TCPClose interface

Functional description The function IL_TCPClose closes a connection created with IL_TCPInitial. This function returns immediately.

If an error occurs, the error code is stored to the variable specified in pudiErrorNum.

Troubleshooting The function uses the error table [ETHERNET_TABLE](#), 16#0150. The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid
COMMUNICATION_ERROR	16#00000211	-	Remote host closed the connection

Tab. 6-28: IL_TCPClose error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.6 UDP asynchronous function blocks

6.6.1 Introduction and overview

The UDP asynchronous function blocks are used in cyclic tasks and have to be called cyclically. Normally, multiple calls are required to complete an operation.

A function block starts with a rising edge at "Enable" and operates until either "Done" or "Error" is TRUE or "Enable" is reset. To start a new operation, reset "Enable" and set it again.

A new UDP socket can be established via the IL_UDPSocketAsync or IL_UDPInitialAsync function blocks.

IL_UDPSocketAsync internally calls the function blocks IL_UDPInitialAsync and IL_UDPCloseAsync and holds the socket handle.

A connection established with IL_UDPInitialAsync can be closed with IL_UDPCloseAsync. If "Server" is set to FALSE when the socket is created, the socket cannot receive any data.



IL_UDPCloseAsync may not be called for sockets established with IL_UDPConnectionAsync.

After the socket was established, data can be exchanged with IL_UDPSendtoAsync and IL_UDPRecvfromAsync.



UDP is a package-based protocol that guarantees neither the order nor the consistency of the data packages transmitted.

Packages can be lost during transmission.

Received data in the received order is handled in a queue.

Each call by IL_UDPRecvfromAsync or IL_UDPRecvfrom returns exactly one package from the queue.

Asynchronous function blocks for a UDP connection:

- [IL_UDPSocketAsync on page 280](#)
- [IL_UDPInitialAsync on page 282](#)
- [IL_UDPSendtoAsync on page 284](#)
- [IL_UDPRecvfromAsync on page 286](#)
- [IL_UDPCloseAsync on page 287](#)

6.6.2 IL_UDPSocketAsync

Brief description The function block IL_UDPSocketAsync establishes either a UDP server or a UDP client depending on the "Server" input.

Interface description

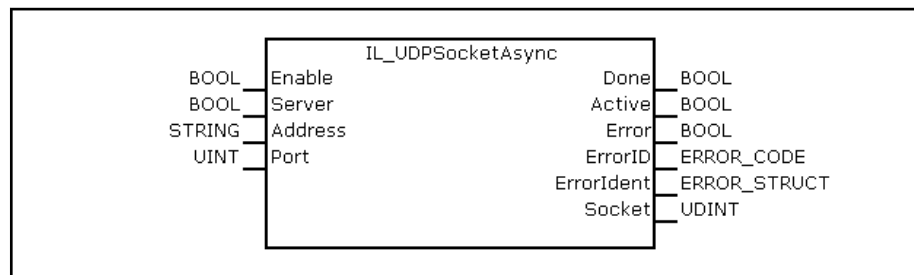


Fig. 6-26: IL_UDPSocketAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Server	BOOL	TRUE: UDP server is created FALSE: UDP client is created
	Address	STRING	If "Server" is set, this input is not used and the own IP address is used. If "Server" is FALSE, this input contains the IP address of the device to be connected.
	Port	UINT	The UDP port at which incoming data packages are expected or to which data packages are to be sent.
VAR_OUTPUT	Done	BOOL	TRUE indicates that the socket was successfully created.
	Active	BOOL	TRUE indicates that the function block creates the socket.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 282)
	Socket	UDINT	Contains the socket handle of the new socket.

Tab. 6-29: IL_UDPSocketAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Server	BOOL			FALSE	Rising edge at "Enable"
Address	STRING			"	Rising edge at "Enable"
Port	UINT	0	65535	0	Rising edge at "Enable"

Tab. 6-30: Min./max. and default values of the IL_UDPSocketAsync function block inputs

Functional description

The function block IL_UDPSocketAsync establishes either a UDP server or a UDP client depending on the "Server" input. Processing starts at a rising edge at "Enable". The function block has to be called cyclically until "Done" or "Error" is TRUE.

In server mode, the function block opens a socket on the UDP port specified in "Port". The socket handle belonging to the socket is displayed at the "Socket" output and the "Done" output is set. While the socket is created, the "Active" output is set.

In client mode, a socket is created but not connected to a port. Accordingly, the port number is unknown and no data can be received. After a successful socket creation, the socket handle is displayed at the "Socket" output and the "Done" output is set. While the socket is created, the "Active" output is set.



In client mode, UDP packages can only be sent but not received.

In case of error, the "Error" output is set. SOCKET_INVALID is displayed at the "Socket" output. The error class and the exact error code can be retrieved at the "ErrorID" and "ErrorIdent" outputs.

If "Enable" is set to FALSE, before "Done" is TRUE, no socket is connected and all internal resources are enabled. If "Enable" is set to FALSE, after the socket was connected, the socket is closed. Possible errors during the connection abort are handled internally and not issued to the error outputs. After "Enable" was set to FALSE, the "Socket" output is set to SOCKET_INVALID.

The "Active" output can be used as a timeout indicator during connection establishment. Typical timeouts for connection establishment range from 2 to 10 seconds.



IL_UDPCloseAsync are not to be used for sockets created with IL_UDPSocketAsync.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting the I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#0000020C	16#00000000	No route to target device

Tab. 6-31: IL_UDPSocketAsync error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.6.3 IL_UDPInitialAsync

Brief description

The function block IL_UDPInitialAsync establishes either a UDP server or a UDP client depending on the "Server" input.

Interface description

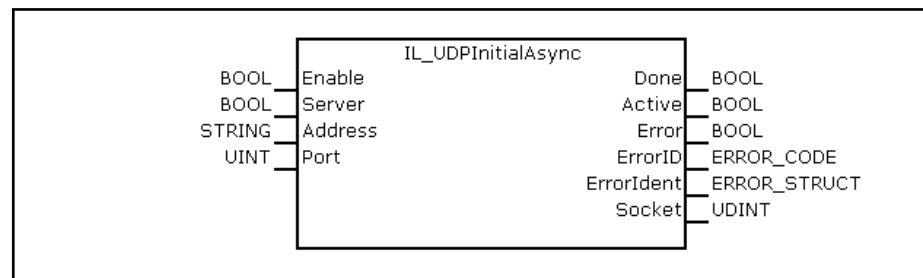


Fig. 6-27: IL_UDPInitialAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Server	BOOL	TRUE: UDP server is created. FALSE: UDP client is created.
	Address	STRING	If "Server" is set, this input is not used and the own IP address is used. If "Server" is FALSE, this input contains the IP address of the device to be connected.
	Port	UINT	The UDP port at which incoming data packages are expected or to which data packages are to be sent.
VAR_OUTPUT	Done	BOOL	TRUE indicates that the socket was successfully created.
	Active	BOOL	TRUE indicates that the function block creates the socket.
	Error	BOOL	TRUE indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 284)
	Socket	UDINT	Contains the socket handle of the new socket.

Tab. 6-32: IL_UDPInitialAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Server	BOOL			FALSE	Rising edge at "Enable"
Address	STRING			"	Rising edge at "Enable"
Port	UINT	0	65535	0	Rising edge at "Enable"

Tab. 6-33: Min./max. and default values of the IL_UDPInitialAsync function block inputs

Functional description

The function block IL_UDPInitialAsync establishes either a UDP server or a UDP client depending on the "Server" input. Processing starts at a rising edge at "Enable". The function block has to be called cyclically until "Done" or "Error" is TRUE.

In server mode, the function block opens a socket on the UDP port specified in "Port". The socket handle belonging to the socket is displayed at the "Socket" output and the "Done" output is set. While the socket is created, the "Active" output is set.

In client mode, a socket is created but not connected to a port. Accordingly, the port number is unknown and no data can be received. After a successful socket creation, the socket handle is displayed at the "Socket" output and the "Done" output is set. While the socket is created, the "Active" output is set.



In client mode, UDP packages can only be sent but not received.

In case of error, the "Error" output is set. SOCKET_INVALID is displayed at the "Socket" output. The error class and the exact error code can be retrieved at the "ErrorID" and "ErrorIdent" outputs.

If "Enable" is set to FALSE, before "Done" is TRUE, no socket is connected and all internal resources are enabled. If "Enable" is set to FALSE, after the socket was applied, the socket is not closed in contrast to IL_UDPSocketAsync. After "Enable" was set to FALSE, the "Socket" output is set to SOCKET_INVALID.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#0000020C	16#00000000	No route to target device

Tab. 6-34: IL_UDPInitialAsync error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.6.4 IL_UDPSendtoAsync

Brief description The function block IL_UDPSendtoAsync sends data packages via a socket created with IL_UDPSocketAsync or IL_UDPInitialAsync.

Interface description

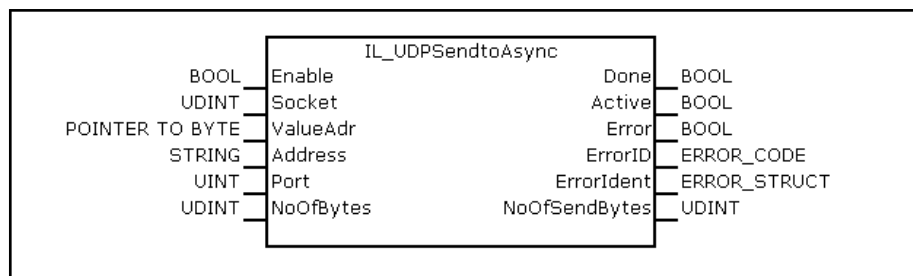


Fig. 6-28: IL_UDPSendtoAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge Binary outputs are reset at a falling edge
	Socket	UDINT	IL_UDPSocketAsync or IL_UDPInitialAsync socket handle
	ValueAdr	POINTER TO BYTE	Initial address of data to be sent
	Address	STRING	Target IP address of the receiver (e.g., '192.168.1.2')
	Port	UINT	UDP reception port of remote host
	NoOfBytes	UDINT	Number of characters to be sent

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	TRUE indicates that the data package was successfully sent
	Active	BOOL	TRUE indicates that the function block sends data
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 285)
	NoOfSend-Bytes	UDINT	Number of bytes sent. 0 indicates that an error occurred or that the socket was closed.

Tab. 6-35: IL_UDPSendtoAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Enable"
Address	STRING			"	Rising edge at "Enable"
Port	UINT	1	65535	0	Rising edge at "Enable"
NoOfBytes	UDINT	0	16#FFFFFFFF	0	Rising edge at "Enable"

Tab. 6-36: Min./max. and default values of the IL_UDPSendtoAsync function block inputs

Functional description

The function block IL_UDPSendtoAsync sends data packages via a socket created with IL_UDPSocketAsync or IL_UDPInitialAsync. Transmission begins at a rising edge at "Enable". The function block has to be called cyclically until "Done" or "Error" is TRUE.

The buffer to be sent is applied to the "ValueAdr" input. The number of characters to be sent is applied at the "NoOfBytes" input.



For some controls, the maximum size of UDP data packages is limited to 1430 bytes.

If "Enable" is reset before "Done" is TRUE, the transmission is interrupted. No data is sent and the "NoOfSentBytes" output is 0.

The "Active" output is set during transmission of the data package. "Active" can be used as a timeout indicator. Typical transmission timeouts range from 1 to 2 seconds. During transmission ("Active" is TRUE), the number of bytes sent can be read at the "NoOfSentBytes" output. Generally, the entire package is sent in one call.

The IP address of the remote host and the target port has to be specified in the "Address" and "Port" inputs.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	ValueAdr is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	NoOfBytes is 0
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid
COMMUNICATION_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found

Tab. 6-37: IL_UDPSendtoAsync error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.6.5 IL_UDPRecvfromAsync

Brief description The function block L_UDPRecvfromAsync sends data packages via a socket created with IL_UDPSocketAsync or IL_UDPInitialAsync.

Interface description

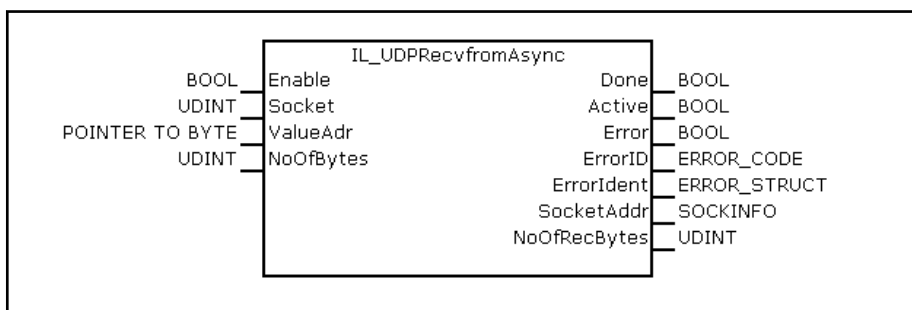


Fig. 6-29: IL_UDPRecvfromAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Socket	UDINT	IL_UDPSocketAsync or IL_UDPInitialAsync socket handle
	ValueAdr	POINTER TO BYTE	Initial address of the received data buffer
	NoOfBytes	UDINT	Size of the received data buffer
VAR_OUTPUT	Done	BOOL	TRUE indicates that the data package was received successfully.
	Active	BOOL	TRUE indicates that the function block is waiting for incoming data packages.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 287)
	SocketAddr	SOCKINFO	IP address and port number of the transmitter
	NoOfRecBytes	UDINT	Number of bytes copied to the reception data buffer 0 indicates that either the socket was closed or an error occurred.

Tab. 6-38: IL_UDPRecvfromAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Enable"
NoOfBytes	UDINT	0	16#FFFFFFFF	0	Rising edge at "Enable"

Tab. 6-39: Min./max. and default values of the IL_UDPRecvfromAsync function block inputs

Functional description

The function block IL_UDPRecvfromAsync sends data packages via a socket created with IL_UDPSocketAsync or IL_UDPInitialAsync.

Transmission begins at a rising edge at "Enable".

The function block has to be called cyclically until "Done" or "Error" is TRUE.

If "Enable" is reset before "Done" is TRUE, the transmission is canceled. In case of such an abort, no data is accepted. "NoOfRecBytes" is hence 0.

The "Active" output is set during data transmission. "Active" can be used as a timeout indicator.

Typical reception timeouts range from 1 to 2 seconds.

Time diagram

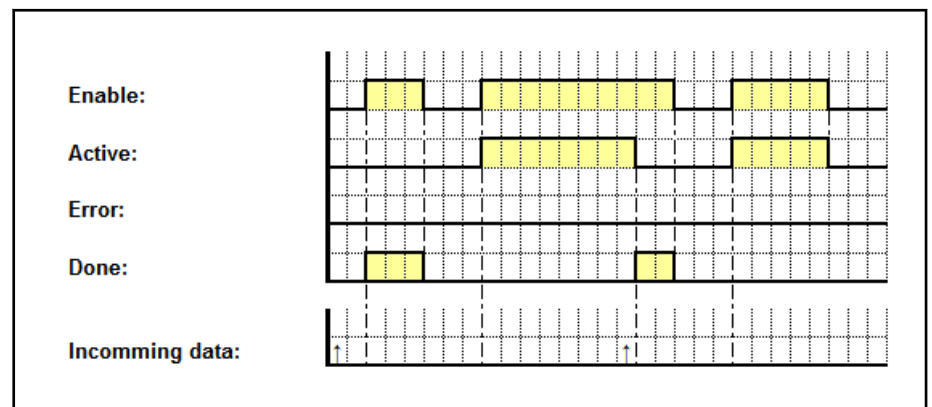


Fig. 6-30: Time diagram of a UDP data package reception

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	ValueAdr is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	NoOfBytes is 0
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid.

Tab. 6-40: IL_UDPRecvfromAsync error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.6.6 IL_UDPCloseAsync

Brief description

The function block IL_UDPCloseAsync closes a socket created with IL_UDPInitialAsync.

Interface description

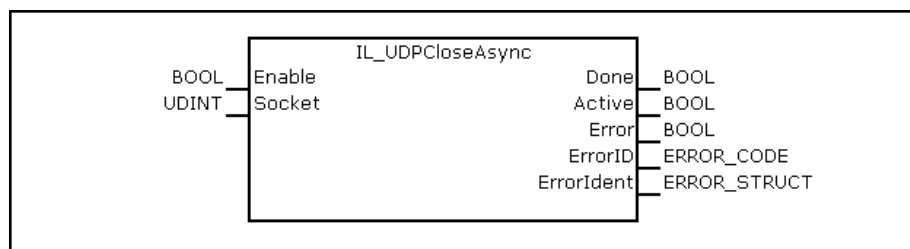


Fig. 6-31: IL_UDPCloseAsync function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge. Binary outputs are reset at a falling edge.
	Socket	UDINT	IL_UDPInitialAsync socket handle
VAR_OUTPUT	Done	BOOL	TRUE indicates that the socket was successfully closed.
	Active	BOOL	TRUE indicates that the socket is closed.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 288)

Tab. 6-41: IL_UDPCloseAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Socket	UDINT	0	16#FFFFFFFF	SOCKET_INVALID	Rising edge at "Enable"

Tab. 6-42: Min./max. and default values of the IL_UDPInitialAsync function block inputs

Functional description

The function block IL_UDPCloseAsync closes a connection created with IL_UDPInitialAsync.

The function block could require more calls than one to terminate the operation. The operation is completed successfully, if the "Done" output is set. "Active" is set while the operation is running. If an error occurs, the "Error" output is set. The outputs "ErrorID" and "ErrorIdent" contain more detailed error information. The function block has to be called after a rising edge at "Enable" until either "Done" or "Error" is set or "Enable" is reset.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00000202	16#00000000	Socket handle invalid.
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection.

Tab. 6-43: IL_UDPCloseAsync error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.7 UDP synchronous functions

6.7.1 Introduction and overview

The synchronous UDP functions are implemented as blocking calls. Thus, they do not return until the operation is completed or an error occurred. These functions are only to be used in free running tasks.

IL_UDPInitial is used to connect a new UDP socket and returns a socket handle.

After the successful creation of the UDP socket, data can be sent and received with IL_UDPSendto and IL_UDPRecvfrom.

If "Server" is set to FALSE when the socket is created, the socket cannot receive any data

IL_UDPClose closes a UDP socket and enables the socket handle.



UDP is a package-based protocol that guarantees neither the order nor the consistency of the data packages transmitted. Packages can be lost during transmission.

Received data in the received order is managed in a queue.

Each call by IL_UDPRecvfromAsync or IL_UDPRecvfrom returns exactly one package from the queue.

Synchronous functions for a UDP connection:

- [IL_UDPInitial on page 289](#)
- [IL_UDPSendto on page 290](#)
- [IL_UDPRecvfrom on page 291](#)
- [IL_UDPClose on page 292](#)

6.7.2 IL_UDPInitial

Brief description

The IL_UDPInitial function establishes either a UDP server or a UDP client depending on the "bServer" parameter.

Interface description

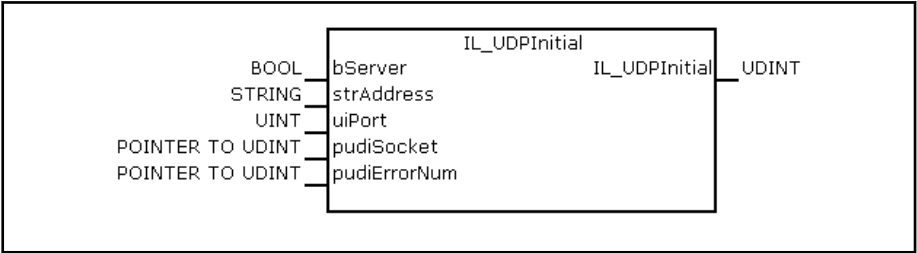


Fig. 6-32: IL_UDPInitial function

	Name	Type	Description
VAR_INPUT	bServer	BOOL	TRUE: UDP server is created. FALSE: UDP client is created.
	strAddress	STRING	If "bServer" is set, this input is not used and the own IP address is used. If "bServer" is FALSE, this input contains the IP address of the device to be connected.
	uiPort	UINT	The UDP port at which incoming data packages are expected or to which data packages are to be sent.
	pudiSocket	POINTER UDINT	TO Contains the socket handle of the new socket.
	pudiErrorNum	POINTER UDINT	TO Contains the error code (see Troubleshooting on page 290).
RETURN		UDINT	0 if the socket was successfully created; otherwise INVALID_SOCKET.

Tab. 6-44: IL_UDPInitial interface

Functional description

The IL_UDPInitial function establishes either a UDP server or a UDP client depending on the "bServer" parameter.

In server mode, the function opens a socket on the UDP port specified in "uiPort". The socket handle belonging to the socket is then displayed in "pudiSocket" output and the function returns with the return value 0.

In client mode, a socket is created but not connected to a port. Accordingly, the port number is unknown and no data can be received. After a successful socket creation, the socket handle is displayed in "pudiSocket" and the function returns with the return value 0.



In client mode, UDP packages can only be sent but not received.

If an error occurs, the function returns with SOCKET_INVALID and stores the error code in "pudiErrorNum".

Troubleshooting

The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	-	Could not create socket
OTHER_ERROR	16#00000041	-	Error when setting I/O mode
OTHER_ERROR	16#00000042	-	Could not connect socket to port
INPUT_INVALID_ERROR	16#00000208	-	Target address invalid
COMMUNICATION_ERROR	16#0000020C	-	No route to target device

Tab. 6-45: IL_UDPInitial error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150 on page 322](#).

6.7.3 IL_UDPSendto

Brief description

The IL_UDPSendto function sends a data package via a socket created with IL_UDPInitial. This function does not return until either all data was transmitted or an error occurred.

Interface description

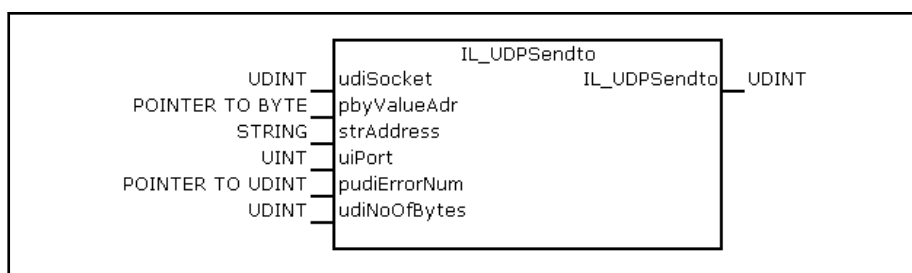


Fig. 6-33: IL_UDPSendto function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_UDPInitial socket handle
	pbyValueAdr	POINTER TO BYTE	Initial address of data to be sent
	strAddress	STRING	Target IP address of the receiver (e.g., '192.168.1.2')
	uiPort	UINT	UDP reception port of remote host
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 291)
	udiNoOfBytes	UDINT	Number of characters to be sent
RETURN		UDINT	Number of bytes actually sent. 0 indicates that either the connection was closed or an error occurred.

Tab. 6-46: IL_UDPSendto interface

Functional description

The IL_UDPSendto function sends a data package via a socket created with IL_UDPInitial. This function does not return until either all data was transmitted or an error occurred.

The buffer to be sent is created in "pbyValueAdr". The number of characters to be sent is transferred in "udiNoOfBytes".

After successful transmission, this function returns the number of characters transmitted. If 0 is returned, the connection was closed before the data packet could be sent. In case of error, the error code is stored to "pudiErrorNum" and the function returns with a return value of 0.

Troubleshooting

The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	-	ValueAdr is not set
INPUT_RANGE_ERROR	16#00000005	-	NoOfBytes is 0
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid
COMMUNICATION_ERROR	16#00000208	-	Target address invalid
COMMUNICATION_ERROR	16#0000020C	-	No network path to target device found

Tab. 6-47: IL_UDPSendto error codes

Other error codes are listed in [ETHERNET_TABLE, ERROR_TABLE 16#0150 on page 322](#).

6.7.4 IL_UDPRecvfrom

Brief description

The function IL_UDPRecvfrom receives data packages via a socket created with IL_UDPInitial. This function does not return until either a data package was received or an error occurred.

Interface description

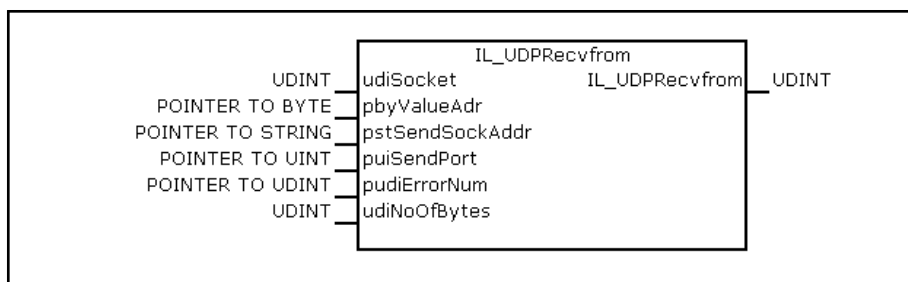


Fig. 6-34: IL_UDPRecvfrom function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_UDPInitial socket handle
	pbyValueAdr	POINTER TO BYTE	Initial address of the received data buffer
	pstrSendSock-Addr	POINTER TO STRING	IP address of the transmitter.
	puiSendPort	POINTER TO UINT	Port number of the transmitter
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 292).
	udiNoOfBytes	UDINT	Size of the received data buffer
RETURN		UDINT	Number of bytes received. 0 indicates that the either the socket was closed or an error occurred.

Tab. 6-48: IL_UDPRecvfrom interface

Functional description

The function IL_UDPRecvfrom receives data packages via a socket created with IL_UDPInitial. This function does not return until either a data package was received or an error occurred.

The reception buffer is created in "pbyValueAdr". The size of the reception buffer is transmitted in "udiNoOfBytes".

After successful transmission, this function returns the number of characters received. If 0 is returned, the socket has been closed before a data package could be received. In case of error, the error code is stored to "pudiErrorNum" and the function returns with a return value of 0.

If no data is received within a second, the function returns with the return value 0 and stores the error code 16#0000020A in "pudiErrorNum".

Troubleshooting

The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	-	ValueAdr is not set
INPUT_RANGE_ERROR	16#00000005	-	NoOfBytes is 0
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid.

Tab. 6-49: IL_UDPRecvfromAsync error codes

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150 on page 322](#).

6.7.5 IL_UDPClose

Brief description

The IL_UDPClose function closes a socket created with IL_UDPInitial. This function returns immediately.

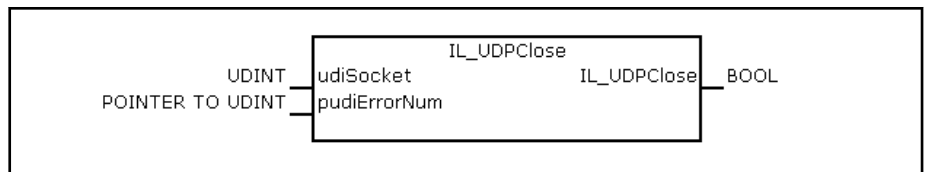
Interface description

Fig. 6-35: IL_UDPClose function

	Name	Type	Description
VAR_INPUT	udiSocket	UDINT	IL_UDPInitial socket handle
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 293).
RETURN		BOOL	TRUE: Socket was closed successfully. FALSE: Error occurred when the socket was closed.

Tab. 6-50: IL_UDPClose interface

Functional description The IL_UDPClose function closes a socket created with IL_UDPInitial. This function returns immediately.

Troubleshooting The function uses the error table ETHERNET_TABLE, 16#0150.
The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR	16#00000202	-	Socket handle invalid.
COMMUNICATION_ERROR	16#00000211	-	Remote host closed the connection.

Tab. 6-51: IL_UDPClose error codes

Other error codes are listed in [ETHERNET_TABLE, ERROR_TABLE 16#0150 on page 322](#).

6.8 Asynchronous ICMP function blocks

6.8.1 IL_PingAsync

Brief description The function block IL_PingAsync performs an ICMP ping to the specified IP address and waits for the response. The ping runtime is displayed at the "EchoTime" output.

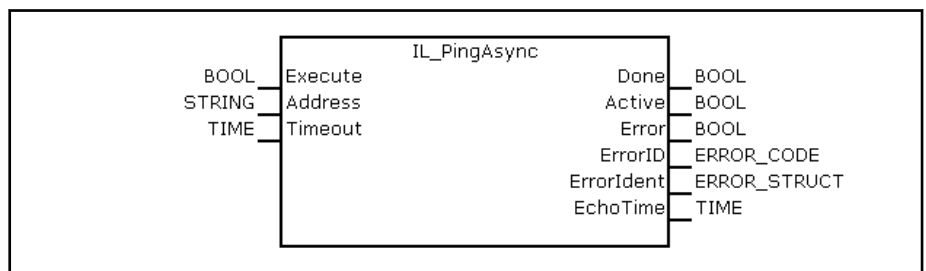
Interface description

Fig. 6-36: IL_PingAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Rising edge starts the ping
	Address	STRING	IP address of the device to be checked
	Timeout	TIME	Maximum ping runtime

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	TRUE: Ping was performed successfully.
	Active	BOOL	TRUE: Ping is currently performed.
	Error	BOOL	An error occurred.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 294)
	EchoTime	TIME	Measured ping runtime

Tab. 6-52: IL_PingAsync interface

Min., max. and default values

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			""	Rising edge at Execute
Timeout	TIME	0 s	N. def.	3 s	Rising edge at Execute

Tab. 6-53: Min., max. and default values of function block IL_PingAsync

Functional description

This function block serves to check the availability of a device in the Ethernet. At a rising edge at "Execute", the IL_PingAsync function block sends a short ICMP ping package to the Ethernet device specified in "Address".

Providing this device is reachable and supports the ICMP protocol, this device returns a ping with the same package content to the transmitter.

The runtime is measured by the function block and issued at the "EchoTime" output.

If the device does not respond within the time specified at the "Timeout" input, an error message is generated at the "Error" outputs. "Active" is set when the function block is waiting for response. Once the response is received, "Done" is set and "EchoTime" contains the runtime.

The "EchoTime" output has the resolution of the calling task. The minimum "EchoTime" of a cyclic 2 ms task is thus 2 ms.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#0000020A	16#00000000	Timeout. No response from the addressed device.
COMMUNICATION_ERROR	16#0000020B	16#00000000	Remote host not found
COMMUNICATION_ERROR	16#0000020C	-	Error code for the system MLD: ERR_NOTIMPLEMENTED: This function is not implemented. Please use the function SysSockPing, library SysSocket

Tab. 6-54: Error codes generated by IL_PingAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150 on page 322](#).

6.9 Synchronous ICMP functions

6.9.1 IL_Ping

Brief description The IL_Ping function sends an ICMP ping to the specified IP address and waits for the response. The function returns the runtime of the ping.

Interface description

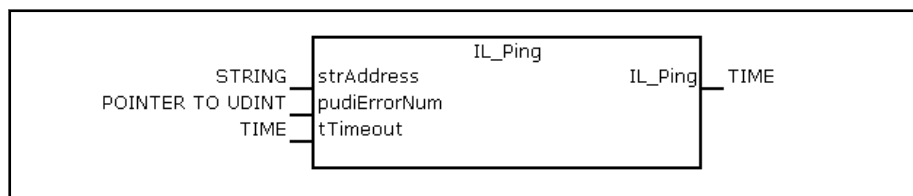


Fig. 6-37: IL_Ping function

	Name	Type	Description
VAR_INPUT	strAddress	STRING	IP address of the device to be checked
	pudiErrorNum	POINTER TO UDINT	Contains the error code (see Troubleshooting on page 295).
	tTimeout	TIME	Maximum ping runtime
RETURN		TIME	Measured ping runtime

Tab. 6-55: IL_Ping interface

Functional description This function checks the availability of a device in the Ethernet.

Upon call, the IL_Ping function sends a short ICMP ping package to the Ethernet device specified in "Address". Providing this device is reachable and supports the ICMP protocol, this device returns a ping with the same package content to the transmitter. The runtime is measured by the function and stored in the return value. If the device does not respond within the time given at the "Timeout" input, an error message is written to "pudiErrorNum".

The function operates synchronously. Thus, it returns when either the device responded, there was a timeout or another error occurred.

Troubleshooting The function uses the error table ETHERNET_TABLE, 16#0150.

The function returns the following error codes in "pudiErrorNum" (Additional1):

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#0000020A	-	Timeout. No response from the addressed device.
COMMUNICATION_ERROR	16#0000020B	-	Remote host not found
COMMUNICATION_ERROR	16#0000020C	-	Error code for the system MLD: ERR_NOTIMPLEMENTED: This function is not implemented. Please use the function SysSockPing, library SysSocket

Tab. 6-56: Error codes generated by IL_Ping

Other error codes are listed in [ETHERNET_TABLE, ERROR_TABLE 16#0150 on page 322](#).

6.10 Asynchronous FTP function blocks

6.10.1 Introduction and overview

FTP (File Transfer Protocol) implements the transfer of individual files via IP. It is based on the TCP protocol.

To transfer a file, an FTP client has to log into an FTP server before. Subsequently, the file can be transferred either from the server to the client (get) or from the client to the server (put). Usually, the FTP server waits for an incoming client connection at the TCP port 21.

Data is transmitted in passive mode. This bypasses possible problems due to firewalls and NAT (Network Address Translation).

If the FTP server does not support the passive mode, active mode is used. Data is always transmitted in binary code. As a result, the line-end identifier is not compiled in texts between Windows- and Unix-based systems.

Asynchronous function blocks for FTP:

- [IL_FTPGetAsync on page 296](#)
- [IL_FTPPutAsync on page 298](#)
- [IL_FTPGetFileAsync on page 301](#)
- [IL_FTPPutFileAsync on page 304](#)

6.10.2 IL_FTPGetAsync

Brief description The function block IL_FTPGetAsync retrieves a file from an FTP server and stores it to a data area.

Interface description

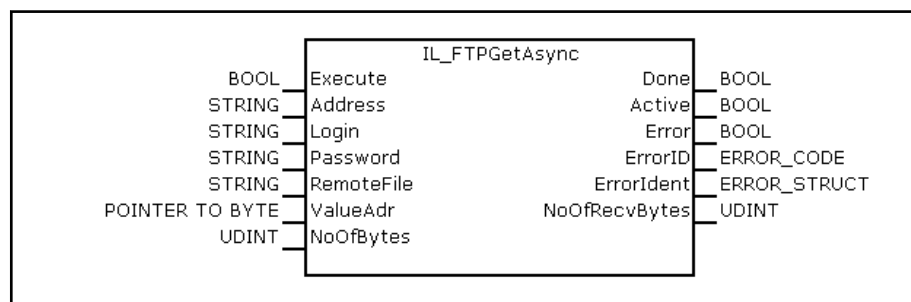


Fig. 6-38: IL_FTPGetAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	Address	STRING	Name or IP address of FTP server
	Login	STRING	Login on FTP server; empty means anonymous access
	Password	STRING	Password for login on FTP server
	RemoteFile	STRING	File name including path on FTP server
	ValueAdr	POINTER TO BYTE	Initial address of the memory area to store data to
	NoOfBytes	UDINT	Size of the memory area specified in "ValueAdr"
VAR_OUTPUT	Done	BOOL	Processing completed without error
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 297)
	NoOfRecvBytes	UDINT	Number of actually transferred bytes.

Tab. 6-57: IL_FTPGetAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Execute"
Login	STRING			"	Rising edge at "Execute"
Password	STRING			"	Rising edge at "Execute"
RemoteFile	STRING			"	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Execute"
NoOfBytes	UDINT	0	65535	0	Rising edge at "Execute"

Tab. 6-58: Min./max. and default values of the inputs of the IL_FTPGetAsync function block

Functional description

The function block IL_FTPGetAsync retrieves a file from an FTP server and stores it to a data area in the RAM of the control.

At a rising edge at "Execute", all inputs are applied and checked for their validity.

Subsequently, the connection is established to the FTP server specified in "Address". The port used is always TCP port 21 registered for FTP.

After the connection was established successfully, "Login" and "Password" are transferred. If the "Login" input is empty, it is tried to login via the "Anonymous" user.

If login was successful, the file transfer is started.

The FTP server transfers the file specified in "RemoteFile" to the storage area specified in "ValueAdr". If the file length exceeds the value specified in "NoOfBytes", the process is canceled. In this case, the storage area at "ValueAdr" does not contain the complete file. A corresponding error message is generated. During transfer, the progress made can be read at the "NoOfRecBytes" output.

Data is always transferred in binary mode. Passive FTP is preferably used for data transfer at which the FTP client establishes a second connection to the FTP server. If this mode is not supported, active FTP is used at which the FTP server establishes a connection to the FTP client for the data transfer.

The function block requires several calls to terminate the operation.

The transfer is completed successfully when the "Done" output is set. "Active" is set during transfer. If an error occurs, the "Error" output is set. The outputs "ErrorID" and "ErrorIdent" contain more detailed error information.



Call the function block after a rising edge at "Execute" until either "Done" or "Error" is set.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
OTHER_ERROR	16#00000040	x	Could not create socket

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not be set to Listen mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
COMMUNICATION_ERROR	16#00002001	16#00000000	Input "RemoteFile" is invalid
COMMUNICATION_ERROR	16#00002002	16#00000000	Target buffer too small for file to be transferred
COMMUNICATION_ERROR	16#00002003	16#00000078	FTP server not ready
COMMUNICATION_ERROR	16#00002003	16#0000014C	Account required for login
COMMUNICATION_ERROR	16#00002003	16#000001A5	Service not ready
COMMUNICATION_ERROR	16#00002003	16#000001A9	Cannot open data connection
COMMUNICATION_ERROR	16#00002003	16#000001AA	Connection closed and file transfer aborted
COMMUNICATION_ERROR	16#00002003	16#000001C2	File not available
COMMUNICATION_ERROR	16#00002003	16#000001C3	Error during file processing on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001C4	Insufficient memory on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001F4	Error in command syntax
COMMUNICATION_ERROR	16#00002003	16#000001F5	Incorrect command parameters
COMMUNICATION_ERROR	16#00002003	16#000001F6	Command not implemented
COMMUNICATION_ERROR	16#00002003	16#000001F7	Incorrect command order
COMMUNICATION_ERROR	16#00002003	16#000001F8	Command with this parameter not implemented
COMMUNICATION_ERROR	16#00002003	16#00000212	Incorrect login or password
COMMUNICATION_ERROR	16#00002003	16#00000226	File not available
COMMUNICATION_ERROR	16#00002003	16#00000227	Unknown page type
COMMUNICATION_ERROR	16#00002003	16#00000228	Insufficient memory
COMMUNICATION_ERROR	16#00002003	16#00000229	File name not allowed
COMMUNICATION_ERROR	16#00002003	16#0000FFFE	Unexpected response from FTP server
COMMUNICATION_ERROR	16#00002003	16#0000FFFF	Unknown response from FTP server

Tab. 6-59: Error codes generated by IL_FTPGetAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.10.3 IL_FTPPutAsync

Brief description The function block IL_FTPPutAsync stores a data area as file to an FTP server.

Interface description

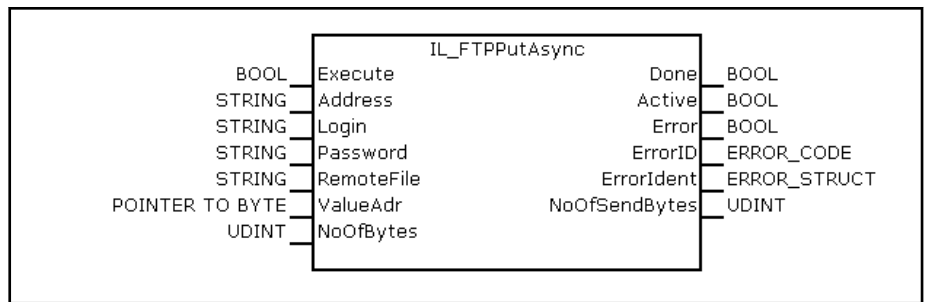


Fig. 6-39: IL_FTPPutAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	Address	STRING	Name or IP address of FTP server
	Login	STRING	Login on FTP server; empty means anonymous access
	Password	STRING	Password for login on FTP server
	RemoteFile	STRING	File name including path on FTP server
	ValueAdr	POINTER TO BYTE	Initial address of data to be sent
	NoOfBytes	UDINT	Number of characters to be transferred
VAR_OUTPUT	Done	BOOL	Processing completed without error
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 300)
	NoOfSend-Bytes	UDINT	Number of actually transferred bytes.

Tab. 6-60: IL_FTPPutAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Execute"
Login	STRING			"	Rising edge at "Execute"
Password	STRING			"	Rising edge at "Execute"
RemoteFile	STRING			"	Rising edge at "Execute"
ValueAdr	POINTER TO BYTE			0	Rising edge at "Execute"
NoOfBytes	UDINT	0	65535	0	Rising edge at "Execute"

Tab. 6-61: Min./max. and default values of the inputs of the IL_FTPPutAsync function block

Functional description

The function block IL_FTPPutAsync transfers data from a data area in the control RAM to an FTP server and stores it as file.

At a rising edge at "Execute", all inputs are applied and checked for their validity. Subsequently, the connection is established to the FTP server specified in "Address".

The port used is always TCP port 21 registered for FTP.

After the connection was established successfully, "Login" and "Password" are transferred. If the "Login" input is empty, it is tried to login via the "Anonymous" user.

If login was successful, the file transfer is started. The number of bytes specified in "NoOfBytes" is transferred from the storage area specified in "ValueAdr" to the FTP server where they are stored as file with the name specified in "RemoteFile". The progress made during transfer can be retrieved at the "NoOfSendBytes" output.

Data is always transferred in binary mode. Passive FTP is preferably used for data transfer at which the FTP client establishes a second connection to the FTP server. If this mode is not supported, active FTP is used at which the FTP server establishes a connection to the FTP client for the data transfer.

The function block requires several calls to terminate the operation. The transfer is completed successfully when the "Done" output is set. "Active" is set during transfer. If an error occurs, the "Error" output is set. The outputs "ErrorID" and "ErrorIdent" contain more detailed error information.



Call the function block after a rising edge at "Execute" until either "Done" or "Error" is set.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not be set to Listen mode.
OTHER_ERROR	16#00000044	x	The incoming connection could not be accepted.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
COMMUNICATION_ERROR	16#00002001	16#00000000	Input "RemoteFile" is invalid
COMMUNICATION_ERROR	16#00002003	16#00000078	FTP server not ready
COMMUNICATION_ERROR	16#00002003	16#0000014C	Account required for login
COMMUNICATION_ERROR	16#00002003	16#000001A5	Service not ready
COMMUNICATION_ERROR	16#00002003	16#000001A9	Cannot open data connection
COMMUNICATION_ERROR	16#00002003	16#000001AA	Connection closed and file transfer aborted
COMMUNICATION_ERROR	16#00002003	16#000001C2	File not available

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#00002003	16#000001C3	Error during file processing on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001C4	Insufficient memory on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001F4	Error in command syntax
COMMUNICATION_ERROR	16#00002003	16#000001F5	Incorrect command parameters
COMMUNICATION_ERROR	16#00002003	16#000001F6	Command not implemented
COMMUNICATION_ERROR	16#00002003	16#000001F7	Incorrect command order
COMMUNICATION_ERROR	16#00002003	16#000001F8	Command with this parameter not implemented
COMMUNICATION_ERROR	16#00002003	16#00000212	Incorrect login or password
COMMUNICATION_ERROR	16#00002003	16#00000226	File not available
COMMUNICATION_ERROR	16#00002003	16#00000227	Unknown page type
COMMUNICATION_ERROR	16#00002003	16#00000228	Not sufficient memory space
COMMUNICATION_ERROR	16#00002003	16#00000229	File name not allowed
COMMUNICATION_ERROR	16#00002003	16#0000FFFE	Unexpected response from FTP server
COMMUNICATION_ERROR	16#00002003	16#0000FFFF	Unknown response from FTP server

Tab. 6-62: Error codes generated by IL_FTPPutAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.10.4 IL_FTPGetFileAsync

Brief description The function block IL_FTPGetFileAsync retrieves a file from an FTP server and stores it on the Flash card.

Interface description

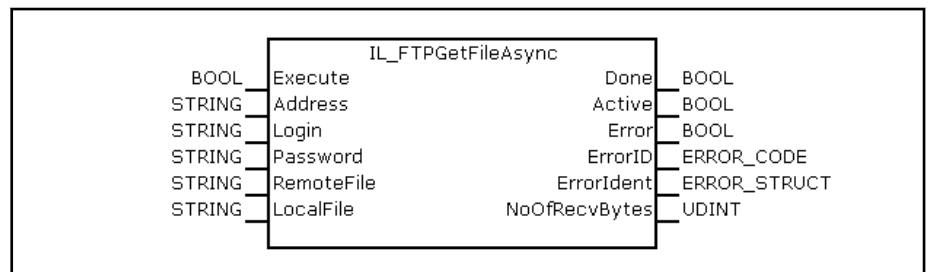


Fig. 6-40: IL_FTPGetFileAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	Address	STRING	Name or IP address of FTP server
	Login	STRING	Login on FTP server; empty means anonymous access
	Password	STRING	Password for login on FTP server
	RemoteFile	STRING	File name including path on FTP server
	LocalFile	STRING	File name including path on Flash card

	Name	Type	Description
VAR_OUTPUT	Done	BOOL	Processing completed without error
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 302)
	NoOfRecBytes	UDINT	Number of actually transferred bytes.

Tab. 6-63: IL_FTPGetFileAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Execute"
Login	STRING			"	Rising edge at "Execute"
Password	STRING			"	Rising edge at "Execute"
RemoteFile	STRING			"	Rising edge at "Execute"
LocalFile	STRING			"	Rising edge at "Execute"

Tab. 6-64: Min./max. and default values of the inputs of the IL_FTPGetFileAsync function block

Functional description

The function block IL_FTPGetAsync retrieves a file from an FTP server and stores it on the Flash card of the control.

At a rising edge at "Execute", all inputs are applied and checked for their validity. Subsequently, the connection is established to the FTP server specified in "Address". The port used is always TCP port 21 registered for FTP.

After the connection was established successfully, "Login" and "Password" are transferred. If the "Login" input is empty, it is tried to login via the "Anonymous" users.

If login was successful, the file transfer is started. The file specified in "RemoteFile" is transferred from the FTP server to the Flash card and stored under the file name specified in "LocalFile". During transfer, the progress made can be read at the "NoOfRecBytes" output.

Data is always transferred in binary mode. Passive FTP is preferably used for data transfer at which the FTP client establishes a second connection to the FTP server. If this mode is not supported, active FTP is used at which the FTP server establishes a connection to the FTP client for the data transfer.

The function block requires several calls to terminate the operation. The transfer is completed successfully when the "Done" output is set. "Active" is set during transfer. If an error occurs, the "Error" output is set. The outputs "ErrorID" and "ErrorIdent" contain more detailed error information.



Call the function block after a rising edge at "Execute" until either "Done" or "Error" is set.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not be set to Listen mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
COMMUNICATION_ERROR	16#00002001	16#00000000	Input "RemoteFile" is invalid
COMMUNICATION_ERROR	16#00002002	16#00000000	Target buffer too small for file to be transferred
COMMUNICATION_ERROR	16#00002003	16#00000078	FTP server not ready
COMMUNICATION_ERROR	16#00002003	16#0000014C	Account required for login
COMMUNICATION_ERROR	16#00002003	16#000001A5	Service not ready
COMMUNICATION_ERROR	16#00002003	16#000001A9	Cannot open data connection
COMMUNICATION_ERROR	16#00002003	16#000001AA	Connection closed and file transfer aborted
COMMUNICATION_ERROR	16#00002003	16#000001C2	File not available
COMMUNICATION_ERROR	16#00002003	16#000001C3	Error during file processing on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001C4	Insufficient memory on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001F4	Error in command syntax
COMMUNICATION_ERROR	16#00002003	16#000001F5	Incorrect command parameters
COMMUNICATION_ERROR	16#00002003	16#000001F6	Command not implemented
COMMUNICATION_ERROR	16#00002003	16#000001F7	Incorrect command order
COMMUNICATION_ERROR	16#00002003	16#000001F8	Command with this parameter not implemented
COMMUNICATION_ERROR	16#00002003	16#00000212	Incorrect login or password
COMMUNICATION_ERROR	16#00002003	16#00000226	File not available
COMMUNICATION_ERROR	16#00002003	16#00000227	Unknown page type
COMMUNICATION_ERROR	16#00002003	16#00000228	Insufficient memory
COMMUNICATION_ERROR	16#00002003	16#00000229	File name not allowed
COMMUNICATION_ERROR	16#00002003	16#0000FFFE	Unexpected response from FTP server
COMMUNICATION_ERROR	16#00002003	16#0000FFFF	Unknown response from FTP server
ACCESS_ERROR	16#00002004	16#00000000	LocalFile could not be created
ACCESS_ERROR	16#00002004	16#00000002	LocalFile could not be written
ACCESS_ERROR	16#00002004	16#00000003	LocalFile could not be closed

Tab. 6-65: Error codes generated by IL_FTPGetFileAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.10.5 IL_FTPPutFileAsync

Brief description The function block IL_FTPPutFileAsync copies a file from a Flash card of the control to an FTP server.

Interface description

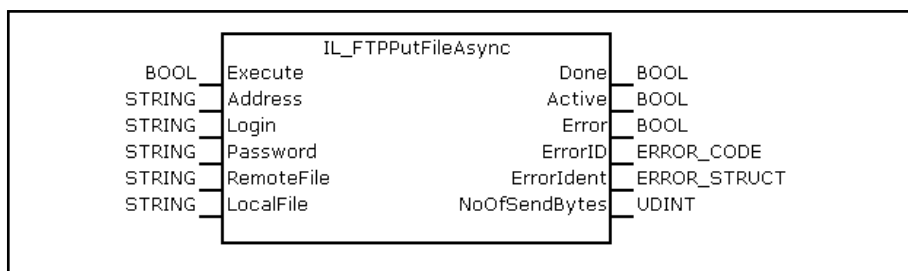


Fig. 6-41: IL_FTPPutFileAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	Address	STRING	Name or IP address of FTP server
	Login	STRING	Login on FTP server; empty means anonymous access
	Password	STRING	Password for login on FTP server
	RemoteFile	STRING	File name including path on FTP server
	LocalFile	STRING	File name including path on Flash card
VAR_OUTPUT	Done	BOOL	Processing completed without error
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 305)
	NoOfSend-Bytes	UDINT	Number of actually transferred bytes.

Tab. 6-66: IL_FTPPutFileAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Execute"
Login	STRING			"	Rising edge at "Execute"
Password	STRING			"	Rising edge at "Execute"
RemoteFile	STRING			"	Rising edge at "Execute"
LocalFile	STRING			"	Rising edge at "Execute"

Tab. 6-67: Min./max. and default values of the inputs of the IL_FTPPutFileAsync function block

Functional description The function block IL_FTPPutFileAsync copies a file from a Flash card of the control to an FTP server.

At a rising edge at "Execute", all inputs are applied and checked for their validity. Subsequently, the connection is established to the FTP server specified in "Address".

The port used is always TCP port 21 registered for FTP.

After the connection was established successfully, "Login" and "Password" are transferred. If the "Login" input is empty, it is tried to login via the "Anonymous" users.

If login was successful, the file transfer is started. The file specified in "LocalFile" is transferred from the Flash card of the control and stored on the FTP server under the file name specified in "RemoteFile". The progress made during transfer can be retrieved at the "NoOfSendBytes" output.

Data is always transferred in binary mode. Passive FTP is preferably used for data transfer at which the FTP client establishes a second connection to the FTP server. If this mode is not supported, active FTP is used at which the FTP server establishes a connection to the FTP client for the data transfer.

The function block requires several calls to terminate the operation. The transfer is completed successfully when the "Done" output is set. "Active" is set during transfer.

If an error occurs, the "Error" output is set. The outputs "ErrorID" and "ErrorIdet" contain more detailed error information.



Call the function block after a rising edge at "Execute" until either "Done" or "Error" is set.

Troubleshooting

The function block uses the error table ETHERNET_TABLE, 16#0150.

It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000004	16#00000000	"ValueAdr" input is not set
INPUT_RANGE_ERROR	16#00000005	16#00000000	"NoOfBytes" input is 0
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not be set to Listen mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
COMMUNICATION_ERROR	16#00002001	16#00000000	Input "RemoteFile" is invalid
COMMUNICATION_ERROR	16#00002003	16#00000078	FTP server not ready
COMMUNICATION_ERROR	16#00002003	16#0000014C	Account required for login
COMMUNICATION_ERROR	16#00002003	16#000001A5	Service not ready
COMMUNICATION_ERROR	16#00002003	16#000001A9	Cannot open data connection
COMMUNICATION_ERROR	16#00002003	16#000001AA	Connection closed and file transfer aborted
COMMUNICATION_ERROR	16#00002003	16#000001C2	File not available

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#00002003	16#000001C3	Error during file processing on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001C4	Insufficient memory on FTP server
COMMUNICATION_ERROR	16#00002003	16#000001F4	Error in command syntax
COMMUNICATION_ERROR	16#00002003	16#000001F5	Incorrect command parameters
COMMUNICATION_ERROR	16#00002003	16#000001F6	Command not implemented
COMMUNICATION_ERROR	16#00002003	16#000001F7	Incorrect command order
COMMUNICATION_ERROR	16#00002003	16#000001F8	Command with this parameter not implemented
COMMUNICATION_ERROR	16#00002003	16#00000212	Incorrect login or password
COMMUNICATION_ERROR	16#00002003	16#00000226	File not available
COMMUNICATION_ERROR	16#00002003	16#00000227	Unknown page type
COMMUNICATION_ERROR	16#00002003	16#00000228	Insufficient memory
COMMUNICATION_ERROR	16#00002003	16#00000229	File name not allowed
COMMUNICATION_ERROR	16#00002003	16#0000FFFE	Unexpected response from FTP server
COMMUNICATION_ERROR	16#00002003	16#0000FFFF	Unknown response from FTP server
ACCESS_ERROR	16#00002004	16#00000000	LocalFile could not be created
ACCESS_ERROR	16#00002004	16#00000001	LocalFile could not be read
ACCESS_ERROR	16#00002004	16#00000003	LocalFile could not be closed

Tab. 6-68: Error codes generated by IL_FTTPutFileAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.11 Asynchronous TFTP function blocks

6.11.1 Introduction and overview

Many peripheral devices use the TFTP protocol (Trivial File Transfer Protocol) to transfer firmware or the configuration. This protocol can be implemented using little resources and is thus used by devices with little memory size.

The function blocks IL_TFTPputFileAsync and IL_TFTPgetFileAsync allow using the TFTP protocol to transfer data from the PLC.

Files can be transferred between the Flash card of the control and the target device.

Both function blocks operate asynchronously. Thus, they do not complete the operation at first call, but have to be called several times until the operation is completed or an error is reported.

Asynchronous function blocks for TFTP:

- [IL_TFTPgetFileAsync](#) on page 306
- [IL_TFTPputFileAsync](#) on page 309

6.11.2 IL_TFTPgetFileAsync

Brief description The function block IL_TFTPgetFileAsync transfers a file from a distant device to a Compact Flash card of the control using the TFTP protocol.

Interface description

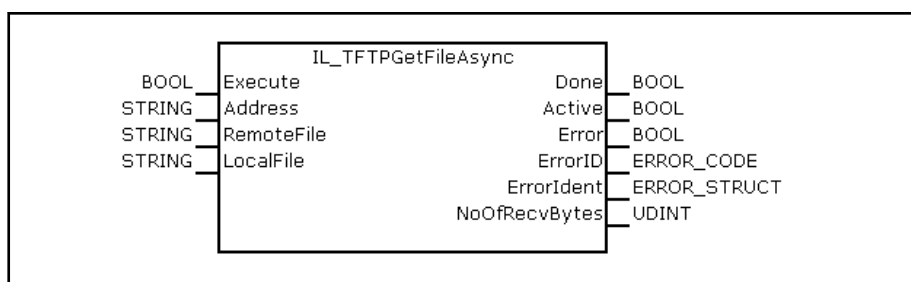


Fig. 6-42: IL_TFTPGetFileAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	Address	STRING	Name or IP address of the target device
	RemoteFile	STRING	File name including path on target device
	LocalFile	STRING	File name including path on Flash card
VAR_OUTPUT	Done	BOOL	Processing completed without error
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 308)
	NoOfRecvBytes	UDINT	Number of actually transferred bytes.

Tab. 6-69: IL_TFTPGetFileAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Execute"
RemoteFile	STRING			"	Rising edge at "Execute"
LocalFile	STRING			"	Rising edge at "Execute"

Tab. 6-70: Min./max. and default values of the inputs of the IL_TFTPGetFileAsync function block

Functional description

The function block IL_TFTPGetFileAsync transfers the file specified in "RemoteFile" from a distant device using the TFTP protocol and stores it on the Compact Flash card of the control under the file name "LocalFile".

Preconditions

The target device has to be reachable from the control via internet protocol (IPv4). This is the case if the device is reachable from the control via ping (see [IL_PingAsync on page 293](#)). However, it is not required to get this explicitly checked by the PLC program. The target device has to support read-only access via TFTP protocol.

Address

The "Address" input contains the IP address of the target device. It has to consist of four decimal numbers between 0 and 255 separated by points as common for the notation in IPv4.

Example:

"192.168.1.42"

RemoteFile

The "RemoteFile" input specifies the file name on the target device. Thus, a specific file can be called by the target device. Not every target device supports the specification of a file name on the target device. In this case, the "RemoteFile" input is ignored.

LocalFile

The "LocalFile" input indicates the file name on the local Compact Flash card of the control. It can also contain a complete path. The default directory is the USER partition of the control.

Time response The IL_TFTPGetFileAsync function block operates asynchronously. After the start due to a rising edge at the "Execute" input, it reads the inputs "Address", "RemoteFile" and "LocalFile".



The function block has to be called cyclically until the successful transfer is signaled via the "Done" output or the "Error" output displays an error. During this period, the "Active" output is set.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
INPUT_INVALID_ERROR	16#00002101	16#00000000	Input "RemoteFile" is invalid
INPUT_INVALID_ERROR	16#00002102	16#00000000	"LocalFile" input is invalid
INPUT_INVALID_ERROR	16#00002103	16#00000000	"LocalFile" not found
ACCESS_ERROR	16#00002104	16#00000000	"LocalFile" could not be written
COMMUNICATION_ERROR	16#00002105	16#00000000	Error while establishing a connection
COMMUNICATION_ERROR	16#00002106	16#00000000	Timeout during transmission
COMMUNICATION_ERROR	16#00002107	16#00000000	Unexpected communication error
ACCESS_ERROR	16#00002107	16#00000001	"RemoteFile" not found
ACCESS_ERROR	16#00002107	16#00000002	Access violation
ACCESS_ERROR	16#00002107	16#00000003	Insufficient memory on FTP server
COMMUNICATION_ERROR	16#00002107	16#00000004	Prohibited TFTP operation
COMMUNICATION_ERROR	16#00002107	16#00000005	Unknown transfer ID
COMMUNICATION_ERROR	16#00002107	16#00000006	File already exists
ACCESS_ERROR	16#00002107	16#00000007	Unknown user
ACCESS_ERROR	16#00002108	16#00000000	"LocalFile" could not be opened.

ErrorID	Additional1	Additional2	Description
ACCESS_ERROR	16#00002108	16#00000002	"LocalFile" could not be written.
ACCESS_ERROR	16#00002108	16#00000003	"LocalFile" could not be closed.

Tab. 6-71: Error codes generated by IL_TFTPGetFileAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.11.3 IL_TFTPPutFileAsync

Brief description The function block IL_TFTPPutFileAsync transfers a file from the Compact Flash card of the control to a distant device using the TFTP protocol.

Interface description

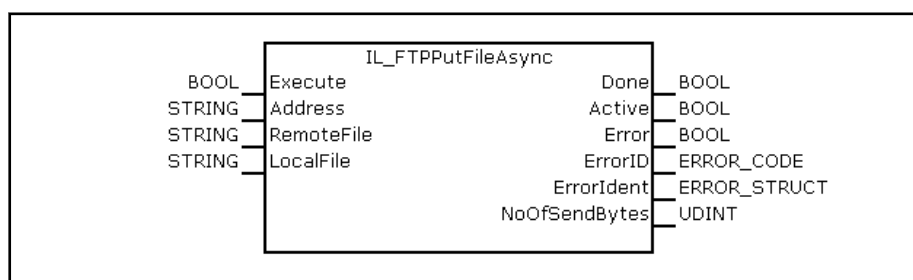


Fig. 6-43: IL_TFTPPutFileAsync function block

	Name	Type	Description
VAR_INPUT	Execute	BOOL	Function block processing enabled (once, edge-controlled)
	Address	STRING	Name or IP address of the target device
	RemoteFile	STRING	File name including path on target device
	LocalFile	STRING	File name including path on Flash card
VAR_OUTPUT	Done	BOOL	Processing completed without error
	Active	BOOL	Processing active
	Error	BOOL	Processing completed with error
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	For detailed error information (refer to Troubleshooting on page 310)
	NoOfSend-Bytes	UDINT	Number of actually transferred bytes.

Tab. 6-72: IL_TFTPPutFileAsync interface

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Execute	BOOL			FALSE	Continuous
Address	STRING			"	Rising edge at "Execute"
RemoteFile	STRING			"	Rising edge at "Execute"
LocalFile	STRING			"	Rising edge at "Execute"

Tab. 6-73: Min./max. and default values of the inputs of the IL_TFTPPutFileAsync function block

Functional description

The function block IL_TFTPputFileAsync transfers the file specified in "LocalFile" from the Compact Flash card of the control to a distant device using the TFTP protocol and stores it under the file name "LocalFile".

Preconditions

The target device has to be reachable from the control via internet protocol (IPv4). This is the case if the device is reachable from the control via ping ([IL_PingAsync on page 293](#)). However, it is not required to get this explicitly checked by the PLC program. The target device has to support write access via TFTP protocol.

Address

The "Address" input contains the IP address of the target device. It has to consist of four decimal numbers between 0 and 255 separated by points as common for the notation in IPv4.

Example:

"192.168.1.42"

RemoteFile

The "RemoteFile" input specifies the file name on the target device. Thus, a specific file can be called by the target device. Not every target device supports the specification of a file name on the target device. In this case, the "RemoteFile" input is ignored.

LocalFile

The "LocalFile" input indicates the file name on the local Compact Flash card of the control. It can also contain a complete path. The default directory is the USER partition of the control.

Time response The IL_TFTPputFileAsync function block operates asynchronously. After the start due to a rising edge at the "Execute" input, it reads the inputs "Address", "RemoteFile" and "LocalFile".



The function block has to be called cyclically until the successful transfer is signaled via the "Done" output or the "Error" output displays an error. During this period, the "Active" output is set.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150. It can generate the following error messages in Additional1/Additional2:

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
INPUT_INVALID_ERROR	16#00002101	16#00000000	Input "RemoteFile" is invalid
INPUT_INVALID_ERROR	16#00002102	16#00000000	"LocalFile" input is invalid
ACCESS_ERROR	16#00002104	16#00000000	"LocalFile" cannot be written
COMMUNICATION_ERROR	16#00002105	16#00000000	Error while establishing a connection
COMMUNICATION_ERROR	16#00002106	16#00000000	Timeout during transmission
COMMUNICATION_ERROR	16#00002107	16#00000000	Unexpected communication error

ErrorID	Additional1	Additional2	Description
ACCESS_ERROR	16#00002107	16#00000001	"RemoteFile" not found
ACCESS_ERROR	16#00002107	16#00000002	Access violation
COMMUNICATION_ERROR	16#00002107	16#00000004	Prohibited TFTP operation
COMMUNICATION_ERROR	16#00002107	16#00000005	Unknown transfer ID
ACCESS_ERROR	16#00002107	16#00000007	Unknown user
ACCESS_ERROR	16#00002108	16#00000000	"LocalFile" could not be opened.
ACCESS_ERROR	16#00002108	16#00000001	"LocalFile" could not be read.
ACCESS_ERROR	16#00002108	16#00000003	"LocalFile" could not be closed.

Tab. 6-74: Error codes generated by IL_TFTPputFileAsync

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.12 Asynchronous communication function blocks for cyclic communication

6.12.1 Introduction and overview

The function blocks under CyclicCommunication allow a cyclic data exchange. They are based on the TCP- and UDP communication function blocks of the RIL_SocketComm.

Data to be exchanged can be specified in its user-defined data structures. Apart from the cyclic communication, data can also be exchanged on request.

The field of application for the cyclic communication is the cyclic data exchange between two controls or another communication partner. Event-controlled communication is required when querying data such as a camera. Both communication methods are supported via UDP and TCP protocol.

Asynchronous function blocks for the cyclic data exchange:

- [IL_TCPCyclic on page 311](#)
- [IL_UDPCyclic on page 316](#)

6.12.2 IL_TCPCyclic

Brief description

The **IL_TCPCyclic** function block sends and receives data via a TCP-based connection.

The cyclic bidirectional operation and the event-controlled transmission mode are supported.

To establish a connection, one communication partner has to be the server, that means that it listens to incoming connections. The other communication partner is the client, that means that it tries to establish a TCP connection to the server. Received packages can be read at every function block call. The function block is enabled at a rising edge at "Enable" and has to be called cyclically. "InOperation" indicates an established connection.

Interface

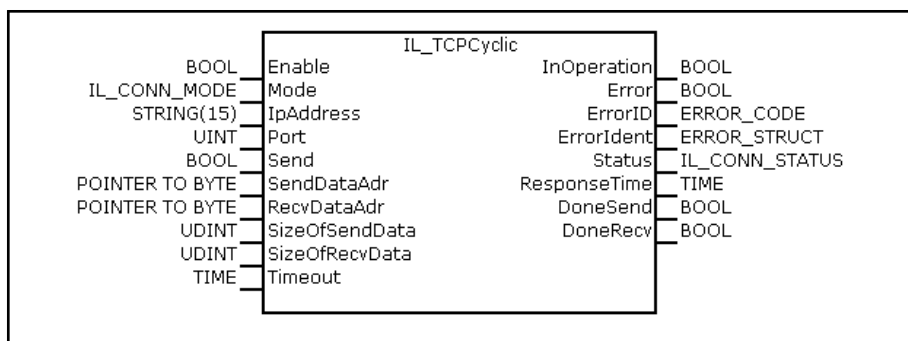


Fig. 6-44: Interface description of the IL_TCPCyclic function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge
	Mode	IL_CONN_MODE	Defines the connection mode, see IL_CONN_MODE , page 320.
	IpAddress	STRING(15)	IP address of the communication partner as aaa.bbb.ccc.ddd
	Port	UINT	Communication port (default: port 50000). The port should be in the range from 50000 to 65535. Communication partners have identical port numbers
	Send	BOOL	Rising edge sends data; only available for the modes "Cyclic_Event_Server", "Cyclic_Event_Client", "Cyclic_Event_Server_Protocol", "Cyclic_Event_Client_Protocol".
	SendDataAdr	POINTER TO BYTE	Pointer to the data to be sent. A data maximum of 1400 bytes can be sent.
	RecvDataAdr	POINTER TO BYTE	Pointer to the data to be received. A data maximum of 1400 bytes can be received.
	SizeOfSendData	UDINT	Length of data to be sent
	SizeOfRecvData	UDINT	Length of the data to be received
	Timeout	TIME	Optional; communication timeout (default: 1500 ms)
VAR_OUTPUT	InOperation	BOOL	TRUE indicates that the function block is ready to send or receive data.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information
	Status	IL_CONN_STATUS	For status information on the function block state, see IL_CONN_STATUS , page 321.
	ResponseTime	TIME	Time required from sending to a successful reception.
	DoneSend	BOOL	TRUE indicates that data was successfully sent for one cycle
	DoneRecv	BOOL	TRUE indicates that data was successfully received for one cycle

Tab. 6-75: Input and outputs of the IL_TCPCyclic function blocks

Min./max. and default values The following table lists the min./max. and default values of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Mode	IL_CONN_MODE	Cyclic_Server	Cyclic_Event_Client_Protocol	Cyclic_Server	Rising edge at "Enable"
IpAddress	STRING(15)	'0.0.0.0'	'255.255.255.255'		Rising edge at "Enable"
Port	UINT	0	65535	50000	Rising edge at "Enable"
Send	BOOL			FALSE	continuously, only available for the modes "Cyclic_Event_Server", "Cyclic_Event_Client", "Cyclic_Event_Server_Protocol", "Cyclic_Event_Client_Protocol"
SendDataAdr	POINTER TO BYTE	>0			Continuous
RecvDataAdr	POINTER TO BYTE	>0			Continuous
SizeOfSendData	UDINT	1	1400		Continuous
SizeOfRecvData	UDINT	1	1400		Continuous
Timeout	TIME			T#1500MS	Continuous

Tab. 6-76: Input behavior of the IL_TCPCyclic

Connection mode

The "Mode" input indicates how the connection is established between two communication partners. This input also determines whether the data is cyclic- or event-controlled and whether the length of the received data is considered.

Possible values are described in the enumeration type [IL_CONN_MODE](#), page 320.

The transmission data length is not considered for the modes "Server_Client", "Cyclic_Client", "Cyclic_Event_Server" and "Cyclic_Event_Client".

If the number of bytes sent from the remote host is higher than the buffer "RecvDataAdr", only "SizeOfRecvData" bytes are copied to the reception buffer "RecvDataAdr".

The remaining and newly received bytes are copied upon next cycle.

The length of the transmission data is considered for the modes "Cyclic_Server_Protocol", "Cyclic_Client_Protocol", "Cyclic_Event_Server_Protocol" and "Cyclic_Event_Client_Protocol".

If the number of bytes sent by the remote host is higher than the configured buffer size, an error is output.

Functional description

The IL_TCPCyclic function block sends and receives data via a TCP/IP-based connection. The order of the characters are sent is always maintained. If an error occurs during transmission, the package in question is automati-

cally resent. Therefore, data cannot get lost. A connection loss is reported in an error code at next read or write access. It is distinguished between an "active" and a "passive" connection:

- The "passive" connection mode specifies a port and then it is waited for queries on this port (so-called "Listening"). This method is used by servers (web server, mail server, etc.).
- For an "active" connection, an IP-address of the remote host and a port is required. This port has to be "open", that means that a client program has to have an established connection on the target device on this port.

After the connection has been established, data can be transmitted in both directions. The function block IL_TCPCyclic establishes a cyclic server/client Ethernet-based network communication between the controls.

Server and client communicate acyclically. Usually, the transferred data (1x there+ 1x back; that is with acknowledgement) is available for further processing within two PLC cycles in the respective PLC program.

Example: The transfer cycle is less than 20 ms for a PLC cycle time of 10 ms. However, data is not available before the second PLC cycle

Supported functions:

- Cyclic communication
- Use as active (client) and passive (server) connection
- Max. telegram length: 1400 bytes
- Event-controlled transmission
- Timeout check
- Check for double telegram failure when receiving or sending
- Check for connection abort

The cyclic communication procedure is as follows:

- The server opens a socket on the port and waits for queries
- The client establishes a connection to the port of the server
- **Cyclic_Client:** The client starts to exchange data.
The client sends a telegram to the server, starts the time monitoring (TimeOut) and waits for the response

Cyclic_Client_Protocol: The client starts to exchange data.

A header with magic number 4711 and data length are send to the server first. The client sends the data, starts the time monitoring (TimeOut) and waits for the response.

- **Cyclic_Server:** The server receives the data from the client.
The server responds with a telegram, starts the time monitoring and waits for the response.

Cyclic_Server_Protocol: The server receives the data from the client.

The header sent by the client is expected first. The magic number 4711 is checked. Subsequently, the server receives the data sent by the client. The data length of the received data is checked against the data length contained in the header. Subsequently, the server responds with a telegram, starts the time monitoring and waits for the response.

- Cyclic change: Server - Client
- A response is expected for each transmitted telegram

The event-controlled communication sequence is as follows:

- The server opens a socket on the port and waits for queries
- The client establishes a connection to the port of the server
- **Cyclic_Event_Client:** If a rising edge is entered at the "Send" input of the Client, the Client sends a telegram to the server and waits for a response.

Cyclic_Event_Client_Protocol: If a rising edge is entered at the "Send" input of the Client, a header with magic number 4711 and the data length is send to the server first. The Client sends the data and waits for a server response

- **Cyclic_Event_Server:** The server receives the data from the client.

In case of a rising edge is entered at the "Send" input, the server responds with a telegram and waits for a response by the client.

Cyclic_Event_Server_Protocol: The server receives the data from the client.

The header sent by the client is expected first. The magic number 4711 is checked. Subsequently, the server receives the data sent by the client. The data length of the received data is checked against the data length contained in the header. In case of a rising edge is entered at the "Send" input, the server responds with a telegram and waits for a response by the client.

Diagnostics The "Status" output indicates the connection state between two communication partners.

Possible values are described in the enumeration type [IL_CONN_STATUS](#), page 321.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150.

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not be set to Listen mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000202	16#00000000	The socket handle is invalid.
INPUT_INVALID_ERROR	16#00000204	16#00000000	The port number is not in a valid range.
COMMUNICATION_ERROR	16#00000207	16#00000000	The port is already assigned.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020A	16#00000000	Timeout occurred.
COMMUNICATION_ERROR	16#0000020B	16#00000000	Remote host not found.
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
COMMUNICATION_ERROR	16#0000020E	16#00000000	The connection was closed.
COMMUNICATION_ERROR	16#0000020F	16#00000000	Connection interrupted by software
INPUT_RANGE_ERROR	16#00000210	16#00000000	Address in SendDataAdr cannot be 0 or lower

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00000210	16#00000001	The size of SizeOfSendData has to be at least 1.
INPUT_RANGE_ERROR	16#00000210	16#00000002	SizeOfSendData must not exceed 1400. A maximum of 1400 bytes can be transmitted.
INPUT_RANGE_ERROR	16#00000210	16#00000003	Address in RecvDataAdr cannot be 0 or lower.
INPUT_RANGE_ERROR	16#00000210	16#00000004	The size of SizeOfRecvData has to be at least 1.
INPUT_RANGE_ERROR	16#00000210	16#00000005	SizeOfRecvData must not exceed 1400. A maximum of 1400 bytes can be received.
INPUT_RANGE_ERROR	16#00000210	16#00000006	SizeOfRecvData is smaller than the amount of the received data
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection.
COMMUNICATION_ERROR	16#00000212	16#00000000	Invalid protocol
COMMUNICATION_ERROR	16#00001001	16#00000000	Data transmission error, sent data does not correspond to the received data
COMMUNICATION_ERROR	16#00001002	16#00000000	Unknown connection mode at the Mode input.

Tab. 6-77: Error codes generated by IL_TCPcyclic

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.12.3 IL_UDPCyclic

Brief description The **IL_UDPCyclic** function block sends and receives data via a UDP-based connection.

The cyclic bidirectional operation and the event-controlled transmission mode are supported.

Interface

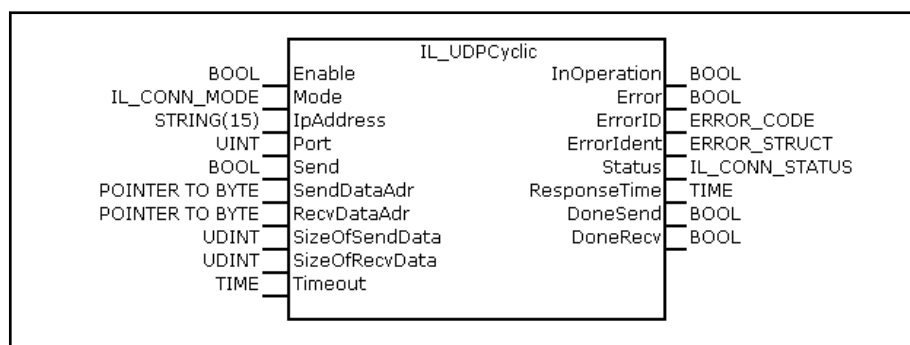


Fig. 6-45: Interface description of the IL_UDPCyclic function block

	Name	Type	Description
VAR_INPUT	Enable	BOOL	Starts at a rising edge
	Mode	IL_CONN_MODE	Defines the connection mode, see IL_CONN_MODE , page 320.
	IpAddress	STRING(15)	IP address of the communication partner as aaa.bbb.ccc.ddd
	Port	UINT	Communication port (default: port 50000). The port should be in the range from 50000 to 65535. The communication partners have an identical port

	Name	Type	Description
	Send	BOOL	Rising edge sends data; only available for the "Cyclic_Event_Server" and "Cyclic_Event_Client" modes.
	SendDataAdr	POINTER TO BYTE	Pointer to the data to be sent. A data maximum of 1400 bytes can be sent.
	RecvDataAdr	POINTER TO BYTE	Pointer to the data to be received. A data maximum of 1400 bytes can be received.
	SizeOfSendData	UDINT	Length of data to be sent
	SizeOfRecvData	UDINT	Length of the data to be received
	Timeout	TIME	Optional; communication timeout (default: 1500 ms)
VAR_OUTPUT	InOperation	BOOL	TRUE indicates that the function block is ready to send or receive data.
	Error	BOOL	TRUE indicates an error.
	ErrorID	ERROR_CODE	Error class
	ErrorIdent	ERROR_STRUCT	Detailed error information
	Status	IL_CONN_STATUS	For status information on the function block state, see IL_CONN_STATUS, page 321
	ResponseTime	TIME	Time required from sending to a successful reception.
	DoneSend	BOOL	TRUE indicates that data was successfully sent for one cycle
	DoneRecv	BOOL	TRUE indicates that data was successfully received for one cycle

Tab. 6-78: Input and outputs of the IL_UDPCyclic function blocks

Min./max. and default values

The following table lists the min./max. and default values of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Mode	IL_CONN_MODE	Cyclic_Server	Cyclic_Event_Client	Cyclic_Server	Rising edge at "Enable"
IpAddress	STRING(15)	'0.0.0.0'	'255.255.255.255'		Rising edge at "Enable"
Port	UINT	0	65535	50000	Rising edge at "Enable"
Send	BOOL			FALSE	Continuous, only for the "Cyclic_Event_Server" and "Cyclic_Event_Client" modes
SendDataAdr	POINTER TO BYTE	>0			Continuous
RecvDataAdr	POINTER TO BYTE	>0			Continuous

Name	Type	Min. value	Max. value	Default value	Effective
SizeOfSendData	UDINT	1	1400		Continuous
SizeOfRecvData	UDINT	1	1400		Continuous
Timeout	TIME			T#1500MS	Continuous

Tab. 6-79: Input behavior of the IL_UDPCyclic

Connection mode

The Mode input indicates how the connection is established between two communication partners. This input also specifies whether data is cyclically transferred or event-controlled.

Possible values are described in the enumeration type [IL_CONN_MODE](#), page 320.

Functional description

The **IL_UDPCyclic** function block sends and receives data via a UDP-based connection.

The cyclic bidirectional operation and the event-controlled transmission mode are supported.

UDP is a connection-free IP protocol. Unlike the TCP protocol, there is no connection establishment prior to the actual data exchange. Communication is package-based. Received data packages are stored in the buffer, but not merged. Every reception call gets exactly one package from the queue. Received UDP data packages are not acknowledged. It is therefore not possible to identify lost packages. UDP is therefore suitable for cyclic data transmission, in which the data of possibly lost packages is updated by later packages.

To communicate, a server has to provide an open port to a client. Received packages can be received and read in one function block call. The function block is enabled at a rising edge at "Enable" and has to be called cyclically.

The server opens a UDP socket at a given port while the client's UDP port is not known. In this configuration, the client can only send but not receive. The server-server variant allows a bidirectional UDP server-server communication. In this configuration, both communication partners can send and receive data packages if the respective target port is specified.

Server and client communicate acyclically for the PLC program. Normally, the transferred data (1x there+ 1x back; that is with acknowledgement) is available for further processing within two PLC cycles in the respective PLC program. Example: The transfer cycle is less than 20 ms for a PLC cycle time of 10 ms. However, data is not available before the second PLC cycle

Supported functions:

- Cyclic communication
- Use as active (client) and passive (server) connection
- Telegram length: 1400 bytes
- Event-controlled transmission
- Timeout check
- Double telegram failure: Checking when receiving or sending
- Connection abort check

The cyclic behavior was achieved by the following measurements:

- The server opens a socket on the port and waits for queries.
- The client connects to the socket on the port on the server

- The client starts the data exchange. It sends a telegram to the server, starts the time monitoring (TimeOut) and waits for the response.
- The server receives data from the client, starts the time monitoring and waits for the response.
- The client receives data from the server, starts the time monitoring and waits for the response.
- Cyclic change: Server - Client as described above
- A response is expected for each sent telegram

Diagnostics The "Status" output indicates the connection state between two communication partners.

Possible values are described in the enumeration type [IL_CONN_STATUS](#), page 321.

Troubleshooting The function block uses the error table ETHERNET_TABLE, 16#0150.

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#00000040	x	Could not create socket
OTHER_ERROR	16#00000041	x	Error when setting I/O mode
OTHER_ERROR	16#00000042	x	Could not connect socket to port
OTHER_ERROR	16#00000043	x	Socket could not be set to Listen mode.
OTHER_ERROR	16#00000044	x	Could not accept incoming connection.
OTHER_ERROR	16#00000045	x	Error when connecting to another device.
INPUT_INVALID_ERROR	16#00000202	16#00000000	The socket handle is invalid.
INPUT_INVALID_ERROR	16#00000204	16#00000000	The port number is not in a valid range.
COMMUNICATION_ERROR	16#00000207	16#00000000	The port is already assigned.
INPUT_INVALID_ERROR	16#00000208	16#00000000	Target address invalid
COMMUNICATION_ERROR	16#00000209	16#00000000	Remote host denied connection establishment
COMMUNICATION_ERROR	16#0000020A	16#00000000	Timeout occurred.
COMMUNICATION_ERROR	16#0000020B	16#00000000	Remote host not found.
COMMUNICATION_ERROR	16#0000020C	16#00000000	No network path to target device found
COMMUNICATION_ERROR	16#0000020E	16#00000000	The connection was closed.
COMMUNICATION_ERROR	16#0000020F	16#00000000	Connection interrupted by software
INPUT_RANGE_ERROR	16#00000210	16#00000000	Address in SendDataAdr cannot be 0 or lower.
INPUT_RANGE_ERROR	16#00000210	16#00000001	The size of SizeOfSendData has to be at least 1.
INPUT_RANGE_ERROR	16#00000210	16#00000002	SizeOfSendData must not exceed 1400. A maximum of 1400 bytes can be transmitted.
INPUT_RANGE_ERROR	16#00000210	16#00000003	Address in RecvDataAdr cannot be 0 or lower.
INPUT_RANGE_ERROR	16#00000210	16#00000004	The size of SizeOfRecvData has to be at least 1.
INPUT_RANGE_ERROR	16#00000210	16#00000005	SizeOfRecvData must not exceed 1400. A maximum of 1400 bytes can be received.
COMMUNICATION_ERROR	16#00000211	16#00000000	Remote host closed the connection.

ErrorID	Additional1	Additional2	Description
COMMUNICATION_ERROR	16#00001001	16#00000000	Data transmission error, sent data does not correspond to the received data
COMMUNICATION_ERROR	16#00001002	16#00000000	Unknown connection mode at the Mode input.

Tab. 6-80: Error codes generated by IL_UDPCyclic

Other error codes are listed in [ETHERNET_TABLE](#), [ERROR_TABLE 16#0150](#) on page 322.

6.12.4 IL_CONN_MODE

The "IL_CONN_MODE " enumeration type indicates how the connection is established between two communication partners.

It also specifies whether data is cyclically transferred or in an event-controlled manner with or without a protocol.

ENUM	Description
Cyclic_Server	Server; cyclic operation, waits for the first telegram from the client
Cyclic_Client	Client; cyclic operation, starts the data exchange
Cyclic_Event_Server	As Cyclic_Server; data can be sent event-controlled via an edge at "Send"
Cyclic_Event_Client	As Cyclic_Client; data can be sent event-controlled via an edge at "Send"
Cyclic_Server_Protocol	Server; cyclic operation, waits for the first telegram from the client. Magic number and transmission data length have to be considered (only available for IL_TCPCyclic , page 311)
Cyclic_Client_Protocol	Client; cyclic operation, starts the data exchange. Magic number and transmission data length are considered (only available for IL_TCPCyclic , page 311)
Cyclic_Event_Server_Protocol	As Cyclic_Server_Protocol; data can be sent event-controlled via an edge at "Send" Magic number and transmission data length are considered (only available for IL_TCPCyclic , page 311)
Cyclic_Event_Client_Protocol	As Cyclic_Client_Protocol; data can be sent event-controlled via an edge at "Send" Magic number and transmission data length are considered (only available for IL_TCPCyclic , page 311)

Tab. 6-81: Connection mode of the function blocks IL_TCPCyclic and IL_UDP-Cyclic



- The communication of the function blocks is bidirectional.
- During reception, the number of bytes specified under "SizeOfRecvData" is expected (applies to Mode = Cyclic_Server, Mode = Cyclic_Client, Mode = Cyclic_Server_Protocol or Mode = Cyclic_Client_Protocol).
- If Mode = Cyclic_Event_*, it is possible to receive less data.
- If different control variants or third-party controls communicate with each other, consider the "natural alignment" and "packed alignment"; that means that it is continuously written in WORD size on the memory range and can thus contain gaps (so-called filling bytes).

Recommendation:

Start with the big data types such as DWORD - WORD - BYTE - BOOL.

STRING is comparable to BYTE

- For the modes "Cyclic_Server_Protocol", "Cyclic_Client_Protocol", "Cyclic_Event_Server_Protocol" and "Cyclic_Event_Client_Protocol":

Header and transmission data are sent/received individually.

At Send:

The header with the two elements "MagicNumber (4711, WORD)" and "DataSize (UINT) = "SizeOfSendData" of the remote host" is sent first.

Subsequently, data is sent.

At Receive:

The header is expected.

The two header elements are checked.

If the MagicNumber (WORD) is unequal to 4711 or if the "DataSize (UINT)" is greater than the input value "SizeOfRecvData", an error is output.

If no error occurs, the data is received.

6.12.5 IL_CONN_STATUS

The enumeration type IL_CONN_STATUS indicates the current connection status between the two communication partners.

ENUM	Description
NotConnected	The function block is disabled. There is no connection
WaitingForConnection	A connection is established or an incoming connection is expected
Running	There is an active communication between server and client (default state in operation; includes the individual states "Receiving", "Received", "Transmitting", "Transmitted")
Receiving	Data is received
Received	Data reception completed
Transmitting	Data is transmitted

ENUM	Description
Transmitted	Data sending completed successfully.
ConnectionTest	Ping connection test performed
Waiting	Waiting state
ReadyToSend	Data can be sent
TimeOutRx	The configured (default = 1500 ms) timeout value was reached
TimeOutTx	The configured (default = 1500 ms) timeout value was reached
NotReadyToSend	Data cannot be sent
DataLengthError	Data package too big. Thus, "SendData" or "RecvData" is bigger than 1400 bytes
ConnectionError	Error in connection. For detailed information, refer to the "RIL_SocketComm.lib" in this documentation
ChecksumError	Incorrect data was transferred. The sent data does not match the received data
DataTransmittingError	Error when sending data. For detailed information, refer to the "RIL_SocketComm.lib" in this documentation
DataReceivingError	Error when receiving data. For detailed information, refer to the "RIL_SocketComm.lib" in this documentation

Tab. 6-82: Diagnostic status of the function blocks IL_TCPcyclic and IL_UDP-Cyclic

6.12.6 ETHERNET_TABLE, ERROR_TABLE 16#0150

ETHERNET_TABLE, General Information

ETHERNET_TABLE - ERROR_TABLE, table number 16#0150

The table contains error numbers output by:

- the function blocks of the RIL_SocketComm library
The RIL_SocketComm library contains the functions and function blocks to establish a connection and for the data transfer via TCP (Transmission Control Protocol) and UDP (User Datagram Protocol).
The RIL_SocketComm library uses the functions of the SysSockets library, Runtime error codes (CmpErrors.library).
- the function block IL_UdpKeys, RIL_HMI_Uilities.library
Function block IL_UdpKeys allows the user to reliably transfer pressing of the keys of an HMI device to the control. The connection is via UDP.
- the function block IL_ModbusTCPClient, RIL_ModbusTCP.library
A Modbus TCP master is implemented in the IL_ModbusTCPClient function block. This function block provides functionality to establish a connection to a ModbusTCP server.

The table was ordered in ascending number of ERROR_STRUCT/Additional1.

ETHERNET_TABLE, General, 16#00000001-16#00000212

Additional1	Additional2	ErrorID	Reason/ solution
16#00000001	16#00000000	STATE_MACHINE_ERROR	Invalid state of the state machine
16#00000004	16#00000000	INPUT_RANGE_ERROR	Input ValueAdr is not set
16#00000005	16#00000000	INPUT_RANGE_ERROR	Input NoOfBytes is 0

Additional1	Additional2	ErrorID	Reason/ solution
16#00000010	16#00000000	INPUT_INVALID_ERROR	Wrong IP syntax
16#00000011	16#00000000	COMMUNICATION_ERROR	Unknown IP address or no connection
16#00000012	16#00000000	DEVICE_ERROR	Invalid internal buffer size
16#00000040	x	OTHER_ERROR	Could not create socket
16#00000041	x	OTHER_ERROR	Could not set IO mode
16#00000042	x	OTHER_ERROR	Could not bind socket
16#00000043	x	OTHER_ERROR	Could not places socket to listening for incoming connection
16#00000044	x	OTHER_ERROR	Could not accept incoming connection attempt on socket
16#00000045	x	OTHER_ERROR	Could not connect to socket
16#00000045	16#00000209	OTHER_ERROR	Establishing a connection was rejected by the remote host
16#00000045	16#0000020A	OTHER_ERROR	Timeout, the remote host does not respond
16#00000045	16#0000020B	OTHER_ERROR	Timeout, remote host not found
16#00000045	16#0000020C	OTHER_ERROR	No route to remote host
16#00000202	16#00000000	INPUT_INVALID_ERROR	Invalid socket handle
16#00000204	16#00000000	INPUT_INVALID_ERROR	Invalid port number
16#00000207	16#00000000	COMMUNICATION_ERROR	Address already in use
16#00000208	16#00000000	INPUT_INVALID_ERROR	Invalid destination address
16#00000209	16#00000000	COMMUNICATION_ERROR	Connection refused
16#0000020A	16#00000000	COMMUNICATION_ERROR	Timeout occurred
16#0000020B	16#00000000	COMMUNICATION_ERROR	Host not found
16#0000020C	16#00000000	COMMUNICATION_ERROR	No route to remote host
16#0000020E	16#00000000	COMMUNICATION_ERROR	Connection closed
16#0000020F	16#00000000	COMMUNICATION_ERROR	Software caused connection abort
16#00000210	16#00000000	COMMUNICATION_ERROR	Data packet oversized. SendDataAdr is not greater than 0
16#00000210	16#00000001	COMMUNICATION_ERROR	SizeOfSendData is not greater than 0
16#00000210	16#00000002	COMMUNICATION_ERROR	DataLength limit of SendDataAdr reached, max. 1400 BYTE
16#00000210	16#00000003	COMMUNICATION_ERROR	Data packet oversized. SendDataAdr is not greater than 0.
16#00000210	16#00000004	COMMUNICATION_ERROR	SizeOfRecvData is not greater than 0
16#00000210	16#00000005	COMMUNICATION_ERROR	DataLength limit of RecvDataAdr reached, max. 1400 BYTE
16#00000210	16#00000006	COMMUNICATION_ERROR	SizeOfRecvData is smaller than size of received Data
16#00000211	16#00000000	COMMUNICATION_ERROR	Connection reset by peer
16#00000212	16#00000000	COMMUNICATION_ERROR	Incompatible protocol

ETHERNET_TABLE, IL_TcpCyclic, IL_UdpCyclic, 16#00001001-16#00001002

Additional1	Additional2	ErrorID	Reason/ solution
16#00001001	16#00000000	COMMUNICATION_ERROR	Data Transmitting Error. SendData not equal to RecvData
16#00001002	16#00000000	COMMUNICATION_ERROR	Wrong Mode selected

ETHERNET_TABLE, IL_FTPGetAsync, IL_FTPPutAsync, IL_FTPGetFileAsync, IL_FTP_Put-FileAsync, 16#00002001-16#00002004

Additional1	Additional2	ErrorID	Reason/ solution
16#00002001	16#00000000	COMMUNICATION_ERROR	Input RemoteFile is invalid
16#00002002	16#00000000	COMMUNICATION_ERROR	Supplied buffer is small for remote file
16#00002003	16#00000078	COMMUNICATION_ERROR	FTP server not ready
16#00002003	16#0000014C	COMMUNICATION_ERROR	Need account for login
16#00002003	16#000001A5	COMMUNICATION_ERROR	Service not ready
16#00002003	16#000001A9	COMMUNICATION_ERROR	Cannot open data connection
16#00002003	16#000001AA	COMMUNICATION_ERROR	Connection closed, transfer aborted
16#00002003	16#000001C2	COMMUNICATION_ERROR	File unavailable
16#00002003	16#000001C3	COMMUNICATION_ERROR	Remote error while processing file
16#00002003	16#000001C4	COMMUNICATION_ERROR	Insufficient storage space on FTP server
16#00002003	16#000001F4	COMMUNICATION_ERROR	Command syntax error
16#00002003	16#000001F5	COMMUNICATION_ERROR	Invalid command parameters
16#00002003	16#000001F6	COMMUNICATION_ERROR	Command not implemented
16#00002003	16#000001F7	COMMUNICATION_ERROR	Bad command sequence
16#00002003	16#000001F8	COMMUNICATION_ERROR	Command not implemented for that parameter
16#00002003	16#00000212	COMMUNICATION_ERROR	Login or password incorrect
16#00002003	16#00000226	COMMUNICATION_ERROR	File unavailable
16#00002003	16#00000227	COMMUNICATION_ERROR	Unknown page type at FTP server
16#00002003	16#00000228	COMMUNICATION_ERROR	Not enough storage space
16#00002003	16#00000229	COMMUNICATION_ERROR	File name not allowed
16#00002003	16#0000FFFE	COMMUNICATION_ERROR	Unexpected FTP server response
16#00002003	16#0000FFFF	COMMUNICATION_ERROR	Unknown FTP server response
16#00002004	16#00000000	ACCESS_ERROR	LocalFile could not be opened
16#00002004	16#00000001	ACCESS_ERROR	LocalFile could not be read
16#00002004	16#00000002	ACCESS_ERROR	LocalFile could not be written
16#00002004	16#00000003	ACCESS_ERROR	LocalFile could not be closed

**ETHERNET_TABLE, IL_TFTPGetFileAsync, IL_TFTPputFileAsync,
16#00002101-16#00002108**

Additional1	Additional2	ErrorID	Reason/ solution
16#00002101	16#00000000	COMMUNICATION_ERROR	Input RemoteFile is invalid
16#00002102	16#00000000	COMMUNICATION_ERROR	Input LocalFile is invalid
16#00002103	16#00000000	COMMUNICATION_ERROR	LocalFile not found
16#00002104	16#00000000	COMMUNICATION_ERROR	Unable to write LocalFile
16#00002105	16#00000000	COMMUNICATION_ERROR	Error while connecting to remote host
16#00002106	16#00000000	COMMUNICATION_ERROR	Timeout while exchanging data
16#00002107	16#00000000	COMMUNICATION_ERROR	Unknown error on remote host
16#00002107	16#00000001	COMMUNICATION_ERROR	Remote file not found
16#00002107	16#00000002	COMMUNICATION_ERROR	Access violation on remote host
16#00002107	16#00000003	COMMUNICATION_ERROR	Insufficient storage space on remote host
16#00002107	16#00000004	COMMUNICATION_ERROR	Invalid TFTP operation
16#00002107	16#00000005	COMMUNICATION_ERROR	Unknown transfer ID
16#00002107	16#00000006	COMMUNICATION_ERROR	RemoteFile does already exist
16#00002107	16#00000007	COMMUNICATION_ERROR	Unknown user
16#00002108	16#00000000	ACCESS_ERROR	LocalFile could not be opened
16#00002108	16#00000001	ACCESS_ERROR	LocalFile could not be read
16#00002108	16#00000002	ACCESS_ERROR	LocalFile could not be written
16#00002108	16#00000003	ACCESS_ERROR	LocalFile could not be closed

7 RIL_LoopControl.library

7.1 Introduction and overview

The RIL_LoopControl provides basic elements and controllers of the control engineering.

The majority of the function blocks are available as time-discrete as well as event-controlled version.

All function blocks are implemented irrespective of the hardware acc. to IEC 61131-3.

The RIL_CommonTypes and RIL_Uilities libraries are a prerequisite.



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

Areas of application and functions/function blocks of RIL_LoopControl

Area of application (folder in library)	
Function blocks	Description
Controllers	
IL_PIDType01, page 331	PID-controller with anti-reset windup integrator and delimiter, time discrete
IL_PIDType02, page 334	PID-controller with anti-reset windup integrator and delimiter, event-controlled
IL_PIDType03, page 338	2DOF (2 degree of freedom) PID-controller with anti-reset windup integrator and delimiter, time discrete
IL_PIDType04, page 342	2DOF (2 degree of freedom) PID-controller with anti-reset windup integrator and delimiter, event-controlled
IL_PIType01, page 346	PI-controller with smooth feedback input, evaluated control deviation, anti-reset windup integrator and delimiter at the output, time discrete
IL_PIType02, page 350	PI-controller with smooth feedback input, evaluated control deviation, anti-reset windup integrator and delimiter at the output, event-controlled
Elements (basic elements)	
IL_DeadtimeType01, page 355	Dead time element (time-variable), time-discrete
IL_DeadtimeType02, page 357	Dead time element (time-variable), event-controlled
IL_DeviationResponseCurve, page 360	Evaluation graph
IL_DT1Type01, page 362	Differential element with first order deceleration, time-discrete
IL_DT1Type02, page 364	Differential element with first order deceleration, event-controlled
IL_PDT1Type01, page 367	Digital filter to increase or decrease the transfer function starting from a certain frequency, time-discrete
IL_PDT1Type02, page 370	Digital filter to increase or decrease the transfer function starting from a certain frequency, event-controlled
IL_PT1Type01, page 373	First order deceleration element, time-discrete
IL_PT1Type02, page 376	First order deceleration element, event-controlled

IL_TransportDelayType01, page 378	Dead time element (runtime-variable), time-discrete
IL_TransportDelayType02, page 381	Dead time element (runtime-variable), event-controlled
IL_ValueLimit, page 384	Value limitation
IL_ValueWindow, page 385	Value range check
Filters	
IL_FIRType01, page 387	Digital FIR filter for high-pass, low-pass and band-pass usage, time discrete
IL_FIRType02, page 389	Digital FIR filter for high-pass, low-pass and band-pass usage, event-controlled
IL_IIRType01, page 391	Digital IIR filter for high-pass, low-pass and band-pass usage, time discrete
IL_IIRType02, page 394	Digital IIR filter for high-pass, low-pass and band-pass usage, event-controlled
IL_StateVariableType01, page 396	IIR filter (Infinite Impulse Response) second degree for high-pass, low-pass and band-pass use and as band-stop filter, time discrete
IL_StateVariableType02, page 398	IIR filter (Infinite Impulse Response) second degree for high-pass, low-pass and band-pass use and as band-stop filter, event-controlled
Generators	
IL_NoiseGeneratorType01, page 402	Pseudorandom number and noise generator, time discrete
IL_NoiseGeneratorType02, page 404	Noise generator for white noise, time discrete
IL_PRBSGeneratorType01, page 405	Generator for pseudorandom binary numbers, time discrete
IL_PRBSGeneratorType02, page 407	Generator for pseudorandom binary numbers, event-controlled
Generators/IL_Signalgenerator	
IL_AngleGeneratorType01, page 410	Angle generator, time discrete
IL_AngleGeneratorType02, page 411	Angle generator, event-controlled
IL_PhaseController, page 412	Phase controller
IL_SignalGeneratorType01, page 414	Signal generator, time discrete
IL_SignalGeneratorType02, page 418	Signal generator, event-controlled
IL_AmplitudeControllerType01, page 422	Amplitude generator, time discrete
IL_AmplitudeControllerType02, page 425	Amplitude generator, event-controlled
Statistics (Statistic function blocks)	
IL_HistogramType01, page 429	Sorts values into categories, time discrete
IL_HistogramType02, page 431	Sorts values into categories, event-controlled
IL_StatisticType01, page 433	Detects floating mean value, minimum value and maximum value, time discrete
IL_StatisticType02, page 435	Detects floating mean value, minimum value and maximum value, event-controlled
IL_SuppressionType01, page 438	suppresses measured values with an excessive deviation to the mean value of the measured values (1 limit), time discrete
IL_SuppressionType02, page 442	suppresses measured values with an excessive deviation to the mean value of the measured values (1 limit), event-controlled

IL_SuppressionType03, page 447	suppresses measured values with an excessive deviation to the mean value of the measured values (2 limits), time discrete
IL_SuppressionType04, page 452	suppresses measured values with an excessive deviation to the mean value of the measured values (2 limits), event-controlled
TemperatureController	
IL_TempControlType01, page 458	Controller for the temperature-controlled system with automatic offline controller setting
IL_TempControlType02, page 476	Controller for the temperature-controlled system with automatic online controller setting
TwoPosCtrl (two-position controller)	
IL_TwoPosCtrlType01, page 489	Two-position controller with hysteresis. It is suitable for simple controls
Utilities (Utilities)	
IL_GetCurrentTimeUs	Documentation in preparation

Tab. 7-1: RIL_LoopControl library

7.2 Time-discrete function blocks

These function blocks calculate a new value with each call if the "Enable" input is active. The time between two calls is measured and is available at the "ActScanTime" output. All calculations refer to the cycle time measured.

Time-discrete function blocks should be called in a cyclic, or even better, a sercos-synchronous task. This can have a significant effect on the accuracy of the output value.

The majority of function block inputs are performed immediately at the next call. It is thus possible to modify control parameters while the system is running.

An active "Pause" input is acknowledged by setting the "InPause" output. The output values are frozen and no more internal calculations take place. "ActScanTime" also continues to be updated when "Pause" is active.

The transfer of a value at the "PresetValue" input to the "ControlValue" or "OutputValue" output takes place at a rising edge at "Preset". Following the transfer, the "PresetAck" output becomes active. If "Preset" is active at the rising edge at "Enable", the value in "PresetValue" is also transferred. An active "Preset" has a higher priority than the "Pause" input.

Time-discrete function blocks can be linked to each other, providing all function blocks are located within the same task. The "InOperation" output of the preceding function block is connected to the "Enable" input of the following function block. In the same way, "InPause" is passed onto "Pause" and "OutputValue" to "Value". The sequence of the linkage has to be taken into account with the sequence of the function block calls.

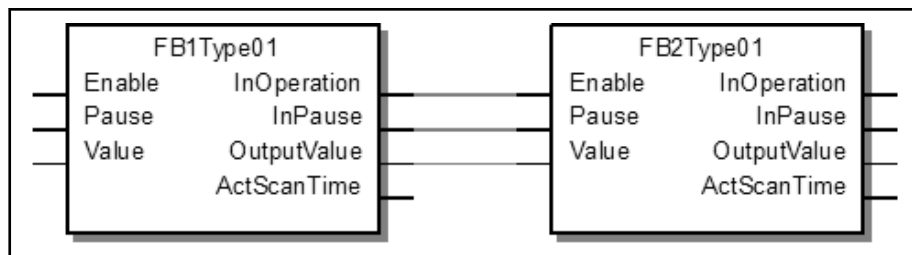


Fig. 7-1: Linking of time-discrete function blocks

7.3 Event-controlled function blocks

These function blocks only calculate one new value if the "Enable" input is active and the value of the "Step" input changes. The last processed step is available at the "ActStep" output. By connecting the "ActStep" output of the preceding function block with the "Step" input of the next block, multiple event-controlled function blocks can be cascaded.

Inputs are only transferred if a change takes place at the "Step" input. This ensures that once a value has been calculated, it cannot be modified at a later stage. A-parameterization while the system is running is thus possible. The "Preset" and "Pause" inputs are active irrespective of a possible change at the "Step" input.

An active "Pause" input is acknowledged by setting the "InPause" output. The output values are frozen and no more internal calculations take place. "Act-Step" is also continues to be updated when "Pause" is active.

The transfer of a value at the "PresetValue" input to the "ControlValue" or "OutputValue" output takes place at a rising edge at "Preset". Following the transfer, the "PresetAck" output becomes active. If "Preset" is active at the rising edge at "Enable", the value in "PresetValue" is also transferred. An active "Preset" has a higher priority than the "Pause" input.

Event-controlled function blocks can be linked directly with each other. The "InOperation" output of the preceding function block is connected to the "Enable" input of the following function block. In the same way, "InPause" is transferred to "Pause", "ActStep" to "Step" and "OutputValue" to "Value". The sequence of the linkage should be taken into account with the sequence of the function block calls. This is not specifically required as the unit of "Act-Step" and "OutputValue" always ensures a correct processing.

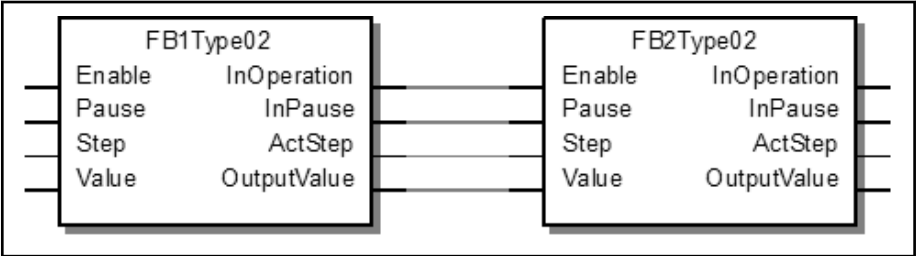


Fig. 7-2: Linking event-controlled function blocks

7.4 Controllers

7.4.1 Introduction and overview

The RIL_LoopControl has the following controllers available:

Areas of application and functions/function blocks of RIL_LoopControl

Area of application (folder in library)	
Function blocks	Description
Controllers	
IL_PIDType01, page 331	PID-controller with anti-reset windup integrator and delimiter, time discrete
IL_PIDType02, page 334	PID-controller with anti-reset windup integrator and delimiter, event-controlled

IL_PIDType03, page 338	2DOF (2 degree of freedom) PID-controller with anti-reset windup integrator and delimiter, time discrete
IL_PIDType04, page 342	2DOF (2 degree of freedom) PID-controller with anti-reset windup integrator and delimiter, event-controlled
IL_PIType01, page 346	PI-controller with smooth feedback input, evaluated control deviation, anti-reset windup integrator and delimiter at the output, time discrete
IL_PIType02, page 350	PI-controller with smooth feedback input, evaluated control deviation, anti-reset windup integrator and delimiter at the output, event-controlled

Tab. 7-2: RIL_LoopControl, controllers

7.4.2 IL_PIDType01

Brief description

The PID-controller contains an anti-reset windup integrator and a delimiter at the output. A preset value can also be set and the controller can be paused. In this case, the controller output remains constant.

The PID-controller function block is to be called cyclically. As it is implemented as digital controller, there can be a significant effect on the accuracy of the controller. The function block measures the time between the calls (scanning time) and calculates all controller parameters if the scanning time deviates substantially or if the controller parameters change. The measured scanning time is also available as an output.

Interface description

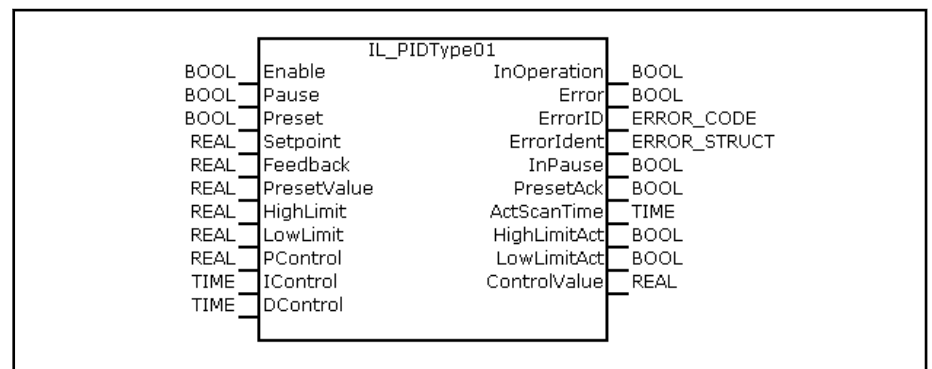


Fig. 7-3: IL_PIDType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "ControlValue" is either set to the preset value of "PresetValue" (if there is a rising edge at "Preset") or to the calculated value
	Pause	BOOL	If TRUE, the value of "ControlValue" is not changed by the controller
	Preset	BOOL	At a rising edge at this input, the controller output is set to the parameterizable value in "PresetValue"
	Setpoint	REAL	The desired "command value" for the controlled process
	Feedback	REAL	The current actual process value
	PresetValue	REAL	Preset value displayed at "ControlValue" at a rising edge at "Preset"
	HighLimit	REAL	Upper limit for "ControlValue" If "ControlValue" exceeds this value, it is limited to this value
	LowLimit	REAL	Lower limit for "ControlValue" If "ControlValue" falls below this value, it is limited to this value

	Name	Type	Comment
	PControl	REAL	Proportional value for the gaining value of the controller.
	IControl	TIME	Integration time "Tn" for the integration value of the controller.
	DControl	TIME	Differential time "Tv" for the differential value of the controller
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PID-controller is working and the "ControlValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 334 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen
	PresetAck	BOOL	The value at "PresetValue" was applied
	ActScan-Time	TIME	The time elapsed since the latest function block call
	HighLimitAct	BOOL	"ControlValue" reached upper limit
	LowLimitAct	BOOL	"ControlValue" reached lower limit
	ControlValue	REAL	Calculated controller output value

Tab. 7-3: IL_PIDType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Preset	BOOL			FALSE	Continuous
Setpoint	REAL	n.def.	n.def.	0.0	Continuous
Feedback	REAL	n.def.	n.def.	0.0	Continuous
PresetValue	REAL	n.def.	n.def.	0.0	Rising edge at "Preset"
HighLimit	REAL	LowLimit	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	HighLimit	0.0	Continuous
PControl	REAL	n.def.	n.def.	0.0	Continuous
IControl	TIME	0 s	n.def.	0 s	Continuous
DControl	TIME	0 s	n.def.	0 s	Continuous

Tab. 7-4: Min./max. and default values of the IL_PIDType01 inputs

PID-controller configuration

The desired P-, I-, PI-, PD- or PID-behavior can be achieved by assigning the corresponding values for the "PControl", "IControl" and "DControl" inputs. The controller parameters can be changed "on-the-fly". Thus, the values are immediately applied when the controller is switched on. The following table can be used to create the required control algorithm.

A specific configuration is produced if all three values for "PControl", "IControl" and "DControl" are set to 0. In this case, the controller is switched off (no P-, I- and D-value) and the "ControlValue" output is fixedly set to 0. However,

the functionality for "Preset" and "Pause" remains. Thus, other values can be forced at the output using these inputs.

Controller type	PControl	IControl	DControl
Switched off	0.0	0 s	0 s
P-controller	Valid value	0 s	0 s
I-controller	0.0	Valid value	0 s
PI-controller	Valid value	Valid value	0 s
PD-controller	Valid value	0 s	Valid value
PID-controller	Valid value	Valid value	Valid value

Tab. 7-5: PID-controller configuration options

The control algorithm can be displayed as the following block diagram:

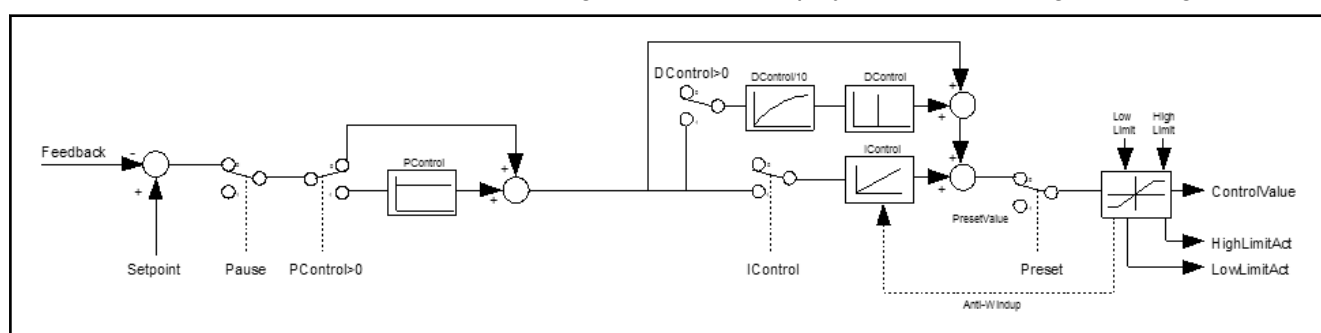


Fig. 7-4: PID-controller block diagram

Functional description

Initialization

During the first action of the function block, the variables $x_{d,k}$, $x_{d,k-1}$, $x_{d,k-2}$, y_k and y_{k-1} are initialized with the value "PresetValue". During initialization, valid values have to be assigned to the inputs "Setpoint" and "Feedback".

Preset value

For a trouble-free start of the controller output "ControlValue", the "Preset" input has to be set if "Enable" is activated. In this case the PID-controller sets the values

$$x_{d,k} = x_{d,k-1} = x_{d,k-2} = \text{Setpoint} - \text{Feedback}$$

and

$$y_k = y_{k-1} = \text{PresetValue}$$

During this time, valid values have to be assigned to the inputs "Setpoint" and "Feedback".

Checking upper/lower limit values is also still active to ensure that the "ControlValue" is limited if the preset value ("PresetValue") is outside the limits of "HighLimit" or "LowLimit".

"PresetAck" becomes TRUE when the value is applied at "PresetValue".

"Preset" has a higher priority than the "Pause" input.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. In this case, the control deviation ("Setpoint" - "Feedback") is set to 0. This causes "ControlValue" to be fixed. It can be used to keep the output constant stable.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Normal mode

If "ControlValue" exceeds or falls below the value of "HighLimit" or "LowLimit", "ControlValue" is limited to the respective value. If the controller has an integration value, an anti-reset wind-up is performed. Thus, the integral value is not added up further. In this case, the "HighLimitAct" and "LowLimitAct" outputs are set.

The controller coefficients are recalculated if controller parameters or scanning time change. Thus, the user the option can fine-tune by adjusting the controller parameters. The scanning time is measured each time the function block is called. If the scanning time deviates by more than 20% from the first one measured, it is assumed that the scanning time changed and all coefficients are recalculated. The function block should only be used in a cyclic task, as the calculations are time-consuming.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000001	"LowLimit" input is greater than the "HighLimit" input value
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000002	Controller type selection is not supported. Check inputs for "PControl", "IControl" and "DControl".
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000002	The "DControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000003	The "IControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000004	The "PControl" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-6: Generated error numbers of the IL_PIDType01 function block

7.4.3 IL_PIDType02

Brief description The PID-controller contains an anti-reset windup integrator and a delimiter at the output. A preset value can also be set and the controller can be paused. In this case, the controller output remains constant.

The PID-controller function block is to be called cyclically. A calculation should only be done if the the value changes at the Step input. The step width is to be kept as constant as possible. The last processed step is also available as an output.

Interface description

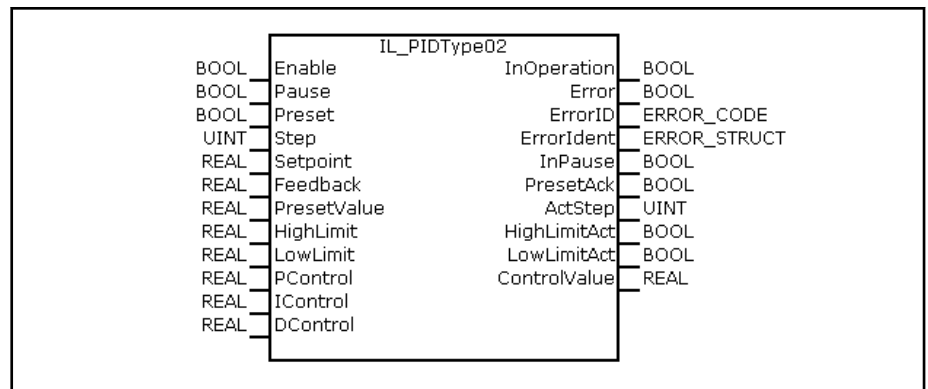


Fig. 7-5: IL_PIDType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "ControlValue" is either set to the preset value of "PresetValue" (if there is a rising edge at "Preset"), or to the calculated value.
	Pause	BOOL	If TRUE, the value of "ControlValue" is not changed by the controller
	Preset	BOOL	At a rising edge at this input, the controller output is set to the parameterizable value in "PresetValue"
	Step	UINT	Processing step
	Setpoint	REAL	The desired "command value" for the controlled process
	Feedback	REAL	The current actual process value
	PresetValue	REAL	A preset value, which is displayed as "ControlValue", if Preset has a rising edge
	HighLimit	REAL	Upper limit for "ControlValue" If "ControlValue" exceeds this value, it is limited to this value
	LowLimit	REAL	Lower limit for "ControlValue" If "ControlValue" falls below this value, it is limited to this value
	PControl	REAL	Proportional value for the gaining value of the controller
	IControl	REAL	Integration constant Tn for the integration value of the controller
	DControl	REAL	Differential constant Tv for the differential value of the controller
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PID-controller operates and the "ControlValue" controller output is valid
	Error	BOOL	Indicates that an error occurred in the function block
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 338 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen
	PresetAck	BOOL	The value at "PresetValue" was applied
	ActStep	UINT	Last processed step
	HighLimitAct	BOOL	"ControlValue" reached upper limit

	Name	Type	Comment
	LowLimitAct	BOOL	"ControlValue" reached lower limit
	ControlValue	REAL	Calculated controller output value

Tab. 7-7: IL_PIDType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Preset	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Setpoint	REAL	n.def.	n.def.	0.0	Continuous
Feedback	REAL	n.def.	n.def.	0.0	Continuous
PresetValue	REAL	n.def.	n.def.	0.0	Rising edge at "Preset"
HighLimit	REAL	LowLimit	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	HighLimit	0.0	Continuous
PControl	REAL	n.def.	n.def.	0.0	Continuous
IControl	REAL	0.0	n.def.	0.0	Continuous
DControl	REAL	0.0	n.def.	0.0	Continuous

Tab. 7-8: Min./max. and default values of the IL_PIDType02 inputs

PID-controller configuration

The desired P, I, PI, PD or PID behavior can be achieved by assigning the corresponding values for the "PControl", "IControl" and "DControl" inputs. The controller parameters can be changed "on-the-fly". Thus, the values are immediately applied when the controller is switched on. The following table can be used to create the required control algorithm.

A specific configuration is produced if all three values for "PControl", "IControl" and "DControl" are set to 0. In this case, the controller is switched off (no P-, I- and D-value) and the "ControlValue" output is fixedly set to 0. However, the functionality for "Preset" and "Pause" remains. Thus, other values can be forced at the output using these inputs.

Controller type	PControl	IControl	DControl
Switched off	0.0	0.0	0.0
P-controller	Valid value	0.0	0.0
I-controller	0.0	Valid value	0.0
PI-controller	Valid value	Valid value	0.0
PD-controller	Valid value	0.0	Valid value
PID-controller	Valid value	Valid value	Valid value

Tab. 7-9: PID-controller configuration options

The control algorithm can be displayed as the following block diagram:

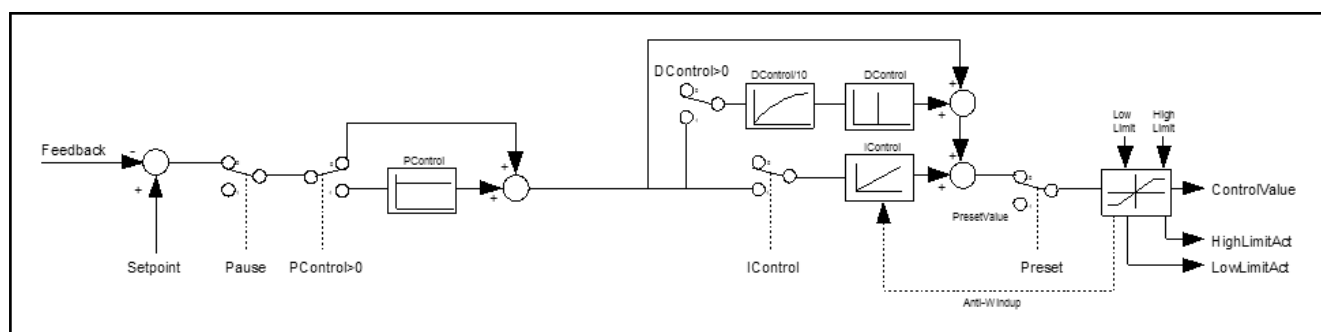


Fig. 7-6: PID-controller block diagram

Functional description**Initialization**

During the first action of the function block, the variables $x_{d,k}$, $x_{d,k-1}$, $x_{d,k-2}$, y_k and y_{k-1} are initialized with the value "PresetValue". During initialization, valid values have to be assigned to the inputs "Setpoint" and "Feedback".

Preset value

If the user wishes the controller output ("ControlValue") to start trouble-free, the "Preset" input has to be set if "Enable" is activated. In this case the PID-controller sets the values

$$x_{d,k} = x_{d,k-1} = x_{d,k-2} = \text{Setpoint} - \text{Feedback}$$

and

$$y_k = y_{k-1} = \text{PresetValue}$$

During this time, valid values have to be assigned to the inputs "Setpoint" and "Feedback".

Checking upper/lower limit values is also still active to ensure that the "ControlValue" is limited if the preset value ("PresetValue") is outside the limits of "HighLimit" or "LowLimit".

"PresetAck" becomes TRUE when the value is applied at "PresetValue".

"Preset" has a higher priority than the "Pause" input.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. In this case, the control deviation ("Setpoint" - "Feedback") is set to 0. This causes "ControlValue" to be fixed. It can be used to keep the output constant stable.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Normal mode

"ControlValue" is only calculated if the "Step" input changes.

If "ControlValue" exceeds or falls below the value of "HighLimit" or "LowLimit", "ControlValue" is limited to the respective value. If the controller has an integration value, an anti-reset wind-up is performed. Thus, the integral value is not added up further. In this case, the "HighLimitAct" and "LowLimitAct" outputs are set.

The controller coefficients are recalculated if the controller parameters or the increment change. Thus, the user the option can fine-tune by adjusting the controller parameters. The increment is measured each time the "Step" input changes. If the increment deviates by more than 20% from the first one measured, all coefficients are recalculated. The function block should only be used in a cyclic task, as the calculations are time-consuming.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000001	"LowLimit" input is greater than the "HighLimit" input value
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000002	Controller type selection is not supported. Check inputs for "PControl", "IControl" and "DControl".
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000002	The "DControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000003	The "IControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000004	The "PControl" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-10: Generated error numbers of the IL_PIDType02 function block

7.4.4 IL_PIDType03

Brief description IL_PIDType03 is the 2DOF (2 degree of freedom) PID-controller.

The control difference is separately created for the proportional, integral and differential value.

The command values for the proportional and differential value are each scaled with the entered scaling factors "bControl" and "cControl". Thus, disturbance and reference variable responses are individually specified. The values "bControl" and "cControl" can be set from 0 to 1.

If both (bControl and cControl) are set to "1.0", the controller responses like a classic 1DOF PID-controller (same response as IL_PIDType01).

The controllers also contain an anti-reset windup integrator and a delimiter at the output. A preset value can also be set and the controller can be paused. In this case, the controller output remains constant.

The PID-controller function block is to be called cyclically. As it is implemented as digital controller, there can be a significant effect on the accuracy of the controller.

The function block measures the time between the calls (scanning time) and calculates all controller parameters if the scanning time deviates substantially or if the controller parameters change. The measured scanning time is also available as an output.

The control coefficients (P-value, I-value and D-value) can be read by the properties "PCorr", "ICorr" and "DCorr".

Interface description

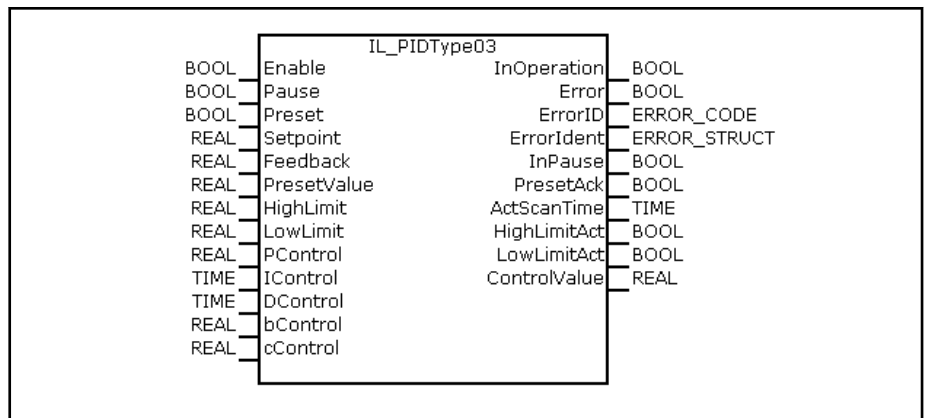


Fig. 7-7: IL_PIDType03 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "ControlValue" is either set to the preset value of "PresetValue" (if there is a rising edge at "Preset") or to the calculated value
	Pause	BOOL	If TRUE, the value of "ControlValue" is not changed by the controller
	Preset	BOOL	At a rising edge at this input, the controller output is set to the parameterizable value in "PresetValue"
	Setpoint	REAL	The desired "command value" for the controlled process
	Feedback	REAL	The current actual process value
	PresetValue	REAL	Preset value displayed at "ControlValue" at a rising edge at "Preset"
	HighLimit	REAL	Upper limit for "ControlValue" If "ControlValue" exceeds this value, it is limited to this value
	LowLimit	REAL	Lower limit for "ControlValue" If "ControlValue" falls below this value, it is limited to this value
	PControl	REAL	Proportional value for the gaining value of the controller
	IControl	TIME	Integration time "Tn" for the integration value of the controller.
	DControl	TIME	Differential time "Tv" for the differential value of the controller
	bControl	REAL	Scaling factor for P-value
	cControl	REAL	Scaling factor for D-value
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PID-controller operates and the "ControlValue" controller output is valid
	Error	BOOL	Indicates that an error occurred in the function block
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error division, see Error codes, page 342
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	PresetAck	BOOL	Value applied at "PresetValue".
	ActScanTime	TIME	The time elapsed since the latest function block call.

	Name	Type	Comment
	HighLimitAct	BOOL	"ControlValue" reached upper limit
	LowLimitAct	BOOL	"ControlValue" reached lower limit
	ControlValue	REAL	Calculated controller output value

Tab. 7-11: IL_PIDType03 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Preset	BOOL			FALSE	Continuous
Setpoint	REAL	n.def.	n.def.	0.0	Continuous
Feedback	REAL	n.def.	n.def.	0.0	Continuous
PresetValue	REAL	n.def.	n.def.	0.0	Rising edge at "Preset"
HighLimit	REAL	LowLimit	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	HighLimit	0.0	Continuous
PControl	REAL	n.def.	n.def.	0.0	Continuous
IControl	TIME	0 s	n.def.	0 s	Continuous
DControl	TIME	0 s	n.def.	0 s	Continuous
bControl	REAL	0	1	1	Continuous
cControl	REAL	0	1	1	Continuous

Tab. 7-12: Min./max. and default values of the IL_PIDType03 inputs

PID-controller configuration

The desired P-, I-, PI-, PD- or PID-behavior can be achieved by assigning the corresponding values for the "PControl", "IControl" and "DControl" inputs. The controller parameters can be changed "on-the-fly". Thus, the values are immediately applied when the controller is switched on. The following table can be used to create the required control algorithm.

A specific configuration is produced if all three values for "PControl", "IControl" and "DControl" are set to 0. In this case, the controller is switched off (no P-, I- and D-value) and the "ControlValue" output is fixedly set to 0. However, the functionality for "Preset" and "Pause" remains. Thus, other values can be forced at the output using these inputs.

Controller type	PControl	IControl	DControl
Switched off	0.0	0 s	0 s
P-controller	Valid value	0 s	0 s
I-controller	0.0	Valid value	0 s
PI-controller	Valid value	Valid value	0 s
PD-controller	Valid value	0 s	Valid value
PID-controller	Valid value	Valid value	Valid value

Tab. 7-13: PID-controller configuration options

The control algorithm can be displayed as the following block diagram:

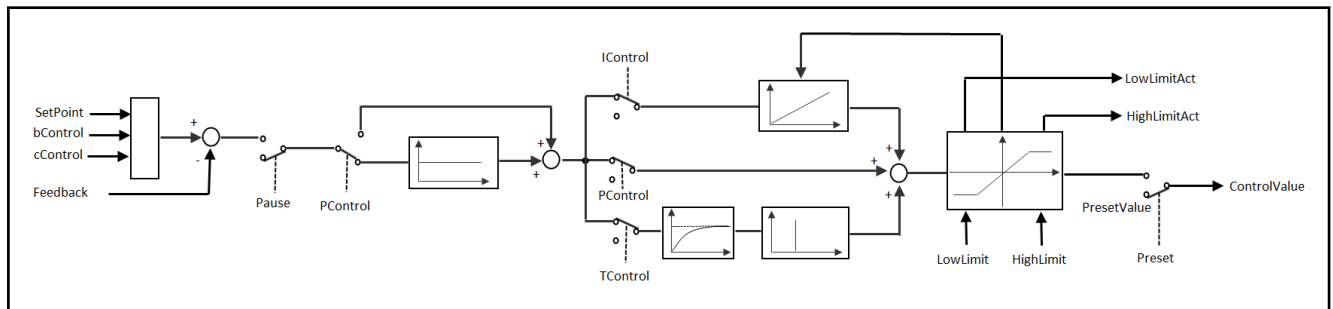


Fig. 7-8: 2DOF PID-controller block diagram

Functional description

Initialization

During initialization, valid values have to be assigned to the inputs "Setpoint" and "Feedback". If the "Preset" input is set to "TRUE", the "ControlValue" output is initialized with the "PresetValue" input. Otherwise, the "ControlValue" output is initialized with "0".

Preset value

For a trouble-free start of the controller output "ControlValue", the "Preset" input has to be set if "Enable" is activated. In this case, the "ControlValue" applies the "PresetValue". During this time, valid values have to be assigned to the inputs "Setpoint" and "Feedback".

Checking upper/lower limit values is also still active to ensure that the "ControlValue" is limited if the preset value ("PresetValue") is outside the limits of "HighLimit" or "LowLimit".

"PresetAck" becomes TRUE when the value is applied at "PresetValue".

"Preset" has a higher priority than the "Pause" input.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. "ControlValue" is fixedly specified in this case. It can be used to keep the output constant stable.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Normal mode

If "ControlValue" exceeds or falls below the value of "HighLimit" or "LowLimit", "ControlValue" is limited to the respective value. In this case, the "HighLimitAct" and "LowLimitAct" outputs are set. If the controller has an integration value, an anti-reset wind-up is performed. Thus, the integral value is not added up further.

The controller coefficients are recalculated if controller parameters or scanning time change. Thus, the user the option can fine-tune by adjusting the controller parameters. The scanning time is measured each time the function block is called. If the scanning time deviates by more than 20% from the first one measured, it is assumed that the scanning time changed and all coefficients are recalculated. The function block should only be used in a cyclic task, as the calculations are time-consuming.

Example on the "bControl" selection

The following figure shows the jump response of a PI-controller with a different "bControl".

The reference behavior of the PI-controller can be changed using "bControl":

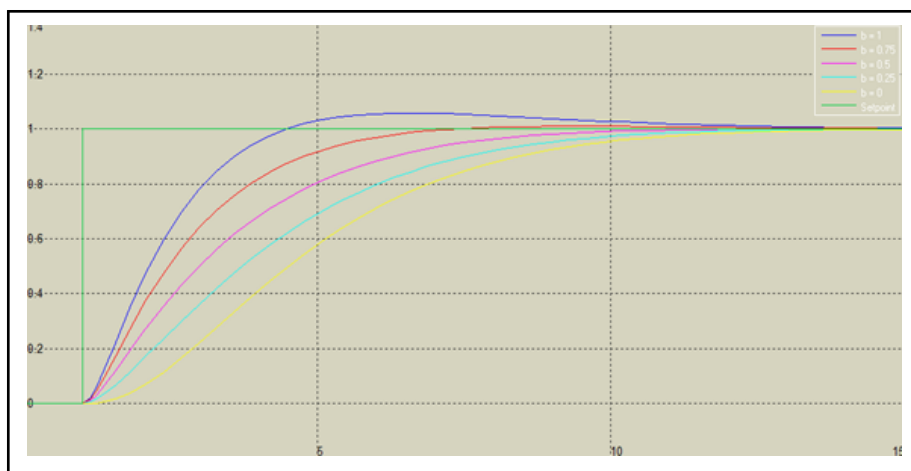


Fig. 7-9: Jump response of a PI-controller with a different "bControl"

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#0C01	16#0001	"LowLimit" input is greater than the "HighLimit" input value
INPUT_INVALID_ERROR, 16#0001	16#0C01	16#0002	Controller type selection is not supported. Check inputs for "PControl", "IControl" and "DControl".
INPUT_RANGE_ERROR, 16#0006	16#0C02	16#0002	The "DControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#0C02	16#0003	The "IControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#0C02	16#0004	The "PControl" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#0006	16#0000	Runtime error in function block: Invalid state of the state machine.
INPUT_INVALID_ERROR, 16#0001	16#0C12	16#0000	The "bControl" input is invalid
INPUT_INVALID_ERROR, 16#0001	16#0C13	16#0000	The "cControl" input is invalid

Tab. 7-14: Generated error numbers of the IL_PIDType03 function block

7.4.5 IL_PIDType04

Brief description IL_PIDType04 is the 2DOF (2 degree of freedom) PID-controller.

The control difference is separately created for the proportional, integral and differential value.

For the proportional and differential value, the command values are individually scaled by the factors "bControl" and "cControl". Thus, disturbance and reference variable responses are individually specified. The values "bControl" and "cControl" can be set from 0 to 1. If both (bControl and cControl) are set to "1.0", the controller responses like a classic 1DOF PID-controller (same response as IL_PIDType02).

The controllers also contain an anti-reset windup integrator and a delimiter at the output. A preset value can also be set and the controller can be paused. In this case, the controller output remains constant.

IL_PIDType04 is a step-dependent controller. The PID-controller function block has to be called cyclically. Only calculate if the value changes at the "Step" input. The step width is to be kept as constant as possible. The last processed step is also available as an output.

The control coefficients (P-value, I-value and D-value) can be read by the properties "PCorr", "ICorr" and "DCorr".

Interface description

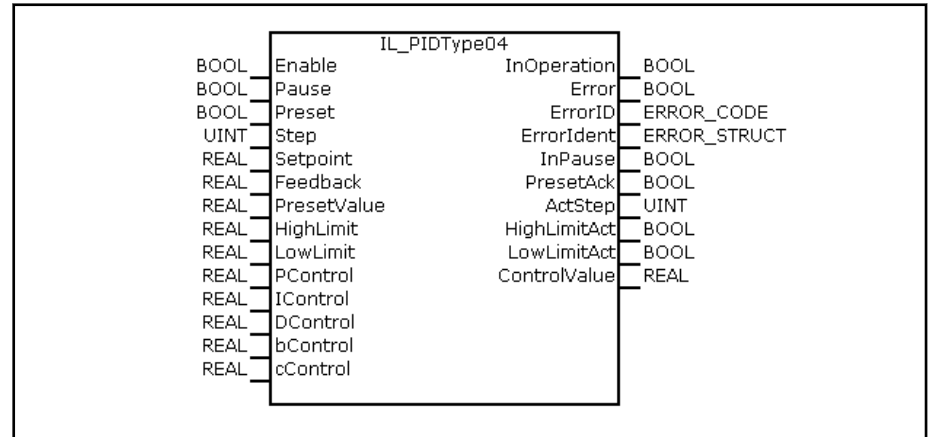


Fig. 7-10: IL_PIDType04 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "ControlValue" is either set to the preset value of "Preset-Value" (if there is a rising edge at "Preset"), or to the calculated value.
	Pause	BOOL	If TRUE, the value of "ControlValue" is not changed by the controller.
	Preset	BOOL	At a rising edge at this input, the controller output is set to the parameterizable value in "PresetValue".
	Step	UINT	Processing step
	Setpoint	REAL	The desired "command value" for the controlled process.
	Feedback	REAL	The current actual value of the process.
	PresetValue	REAL	Preset value displayed as "ControlValue", in case of a rising edge at "Preset".
	HighLimit	REAL	Upper limit for "ControlValue". If "ControlValue" exceeds this value, it is limited to this value.
	LowLimit	REAL	Lower limit for "ControlValue". If "ControlValue" falls below this value, it is limited to this value.
	PControl	REAL	Proportional value for the gaining value of the controller.
	IControl	REAL	Integration time "Tn" for the integration value of the controller.
	DControl	REAL	Differential constant "Tv" for the differential value of the controller.
	bControl	REAL	Scaling factor for P-value.
	cControl	REAL	Scaling factor for D-value.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PID-controller operates and the "ControlValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.

	Name	Type	Comment
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error division, see Error codes, page 346
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	PresetAck	BOOL	Value applied at "PresetValue".
	ActStep	UINT	Last processed step.
	HighLimitAct	BOOL	"ControlValue" reached upper limit
	LowLimitAct	BOOL	"ControlValue" reached lower limit
	ControlValue	REAL	Calculated controller output value

Tab. 7-15: IL_PIDType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Preset	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Setpoint	REAL	n.def.	n.def.	0.0	Continuous
Feedback	REAL	n.def.	n.def.	0.0	Continuous
PresetValue	REAL	n.def.	n.def.	0.0	Rising edge at "Preset"
HighLimit	REAL	LowLimit	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	HighLimit	0.0	Continuous
PControl	REAL	n.def.	n.def.	0.0	Continuous
IControl	REAL	0.0	n.def.	0.0	Continuous
DControl	REAL	0.0	n.def.	0.0	Continuous
bControl	REAL	0.0	1.0	1.0	Continuous
cControl	REAL	0.0	1.0	1.0	Continuous

Tab. 7-16: Min./max. and default values of the IL_PIDType04 inputs

PID-controller configuration

The desired P-, I-, PI-, PD- or PID-behavior can be achieved by assigning the corresponding values for the "PControl", "IControl" and "DControl" inputs. The controller parameters can be changed "on-the-fly". Thus, the values are immediately applied when the controller is switched on. The following table can be used to create the required control algorithm.

There is a special configuration if all the three values for "PControl", "IControl" and "DControl" are set to "0". In this case, the controller is switched off (no P-, I- and D-value) and the "ControlValue" output is fixedly set to 0. However, the functionality for "Preset" and "Pause" remains. Thus, other values can be forced at the output using these inputs.

Controller type	PControl	IControl	DControl
Switched off	0.0	0.0	0.0
P-controller	Valid value	0.0	0.0
I-controller	0.0	Valid value	0.0
PI-controller	Valid value	Valid value	0.0
PD-controller	Valid value	0.0	Valid value
PID-controller	Valid value	Valid value	Valid value

Tab. 7-17: PID-controller configuration options

The control algorithm can be displayed as the following block diagram:

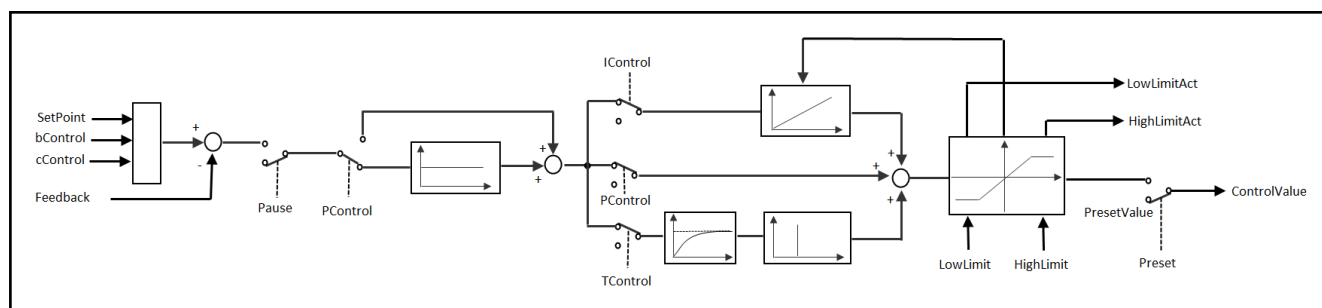


Fig. 7-11: 2DOF PID-controller block diagram

Functional description

Initialization

During initialization, valid values have to be assigned to the inputs "Setpoint" and "Feedback". If the "Preset" input is set to "TRUE", the "ControlValue" output is initialized with the "PresetValue" input. Otherwise, the "ControlValue" output is initialized with "0".

Preset value

For a trouble-free start of the controller output "ControlValue", the "Preset" input has to be set if "Enable" is activated. In this case, the "ControlValue" applies the "PresetValue". During this time, valid values have to be assigned to the inputs "Setpoint" and "Feedback". Checking upper/lower limit values is also still active to ensure that the "ControlValue" is limited if the preset value ("PresetValue") is outside the limits of "HighLimit" or "LowLimit".

"PresetAck" becomes TRUE when the value is applied at "PresetValue".

"Preset" has a higher priority than the "Pause" input.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. "ControlValue" is fixedly specified in this case. It can be used to keep the output constant stable. While pausing, the control coefficients are internally updated.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Normal mode

"ControlValue" is only calculated if the "Step" input changes. If "ControlValue" exceeds or falls below the value of "HighLimit" or "LowLimit", "ControlValue" is limited to the respective value. If the controller has an integration value, an anti-reset wind-up is performed. Thus, the integral value is not added up further. In this case, the "HighLimitAct" and "LowLimitAct" outputs are set.

The controller coefficients are recalculated if the controller parameters or the increment change. Thus, the user the option can fine-tune by adjusting the

Example on the "bControl" selection

controller parameters. The increment is measured each time the "Step" input changes.

The following figure shows the jump response of a PI-controller with a different "bControl".

The reference behavior of the PI-controller can be changed using "bControl":

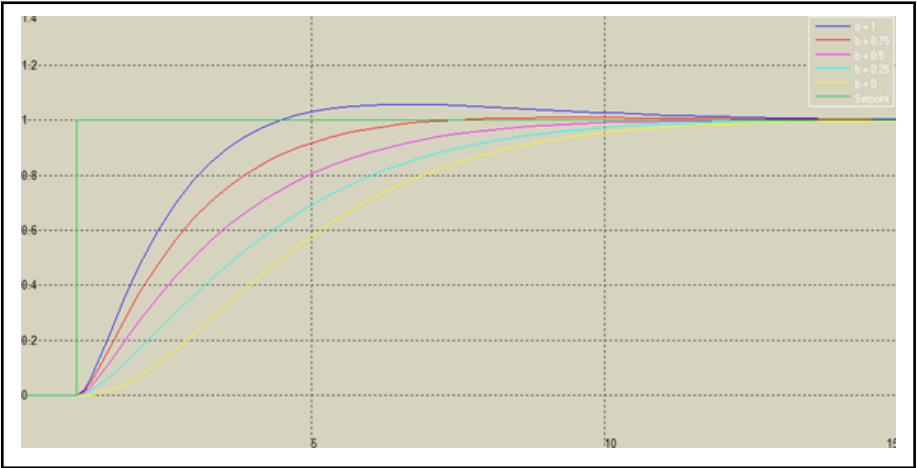


Fig. 7-12: Jump response of a PI-controller with a different "bControl"

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000001	"LowLimit" input is greater than the "HighLimit" input value
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000002	Controller type selection is not supported. Check inputs for "PControl", "IControl" and "DControl".
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000002	The "DControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000003	The "IControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000004	The "PControl" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.
INPUT_INVALID_ERROR, 16#0001	16#00000C12	16#00000000	The "bControl" input is invalid
INPUT_INVALID_ERROR, 16#0001	16#00000C13	16#00000000	The "cControl" input is invalid

Tab. 7-18: Generated error numbers of the IL_PIDType04 function block

7.4.6 IL_PIType01

Brief description

The PI-controller contains an anti-reset windup integrator and a delimiter at the output. The actual value input is filtered by an adjustable PT1-element. The difference between the set value and the actual value is evaluated by a set point curve. A preset value can also be set and the controller can be paused. In this case, the controller output remains constant.

The PI-controller function block should be called cyclically. As it is implemented as digital controller, there can be a significant effect on the accuracy of the controller. The function block measures the time between the calls (scan-

ning time) and calculates all controller parameters if the scanning time deviates substantially or if the controller parameters change. The measured scanning time is available as an output.

Interface description

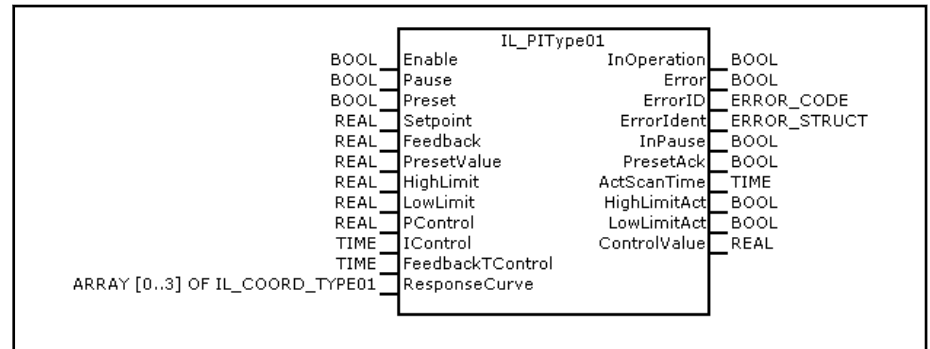


Fig. 7-13: IL_PType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "ControlValue" is either set to the preset value of "PresetValue" (if a rising edge is set at "Preset"), or to the calculated value.
	Pause	BOOL	If TRUE, the value of "ControlValue" is not changed by the controller.
	Preset	BOOL	At a rising edge at this input, the controller output is set to the parameterizable value in "PresetValue".
	Setpoint	REAL	The desired command or reference value for the controlled process.
	Feedback	REAL	The current actual value of the process.
	PresetValue	REAL	A preset value which is displayed at "ControlValue" when there is a rising edge at "Preset".
	HighLimit	REAL	Upper limit for "ControlValue". If "ControlValue" exceeds this value, it is limited to this value.
	LowLimit	REAL	Lower limit for "ControlValue". If "ControlValue" falls below this value, it is limited to this value.
	PControl	REAL	Proportional value for the gaining value of the controller.
	IControl	TIME	Integration time "Tn" for the integration value of the controller.
VAR_OUTPUT	FeedbackT-Control	TIME	PT1 element filter time constant, which smoothes the "Feedback" actual value input.
	Response-Curve	ARRAY[0..3] OF IL_COORD_TYPE01	Setpoint curve for evaluating the difference between the smoothed "Feedback" actual value and the "Setpoint" value. IL_COORD_TYPE01, page 491
	InOperation	BOOL	If TRUE, then the PI-controller works and the "ControlValue" controller output is valid
	Error	BOOL	Indicates that an error occurred in the function block
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 350 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	PresetAck	BOOL	Value at "PresetValue" was applied

	Name	Type	Comment
	ActScan-Time	TIME	The time elapsed since the latest function block call
	HighLimitAct	BOOL	"ControlValue" reached upper limit
	LowLimitAct	BOOL	"ControlValue" reached lower limit
	ControlValue	REAL	Calculated controller output value

Tab. 7-19: IL_PType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Preset	BOOL			FALSE	Continuous
Setpoint	REAL	n.def.	n.def.	0.0	Continuous
Feedback	REAL	n.def.	n.def.	0.0	Continuous
PresetValue	REAL	n.def.	n.def.	0.0	Rising edge at Preset
HighLimit	REAL	LowLimit	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	HighLimit	0.0	Continuous
PControl	REAL	n.def.	n.def.	0.0	Continuous
IControl	TIME	0 s	n.def.	0 s	Continuous
FeedbackTControl	TIME	0 s	n.def.	0 s	Continuous
ResponseCurve	ARRAY [0..3] OF IL_COORD_TYPE01			0	Rising edge at Enable

Tab. 7-20: Min./max. and default values of the IL_PType01 inputs

Controller configuration

The signal at the "Feedback" input is smoothed by a delay element of the first order. The associated filter time constant is set via "FeedbackTControl". If "FeedbackTControl" is 0s, the filter is deactivated. The difference signal from filtered "Feedback" input and "Setpoint" is scaled by a characteristic curve given in "ResponseCurve". The characteristic curve is point-symmetrical to the origin. The x-values of the characteristic curve correspond to the differential signal. The y-values correspond to the scaled difference signal. Up to four points can be specified. The sections between the points are linearly interpolated. If "ResponseCurve" is not applied, there is no scaling.

The desired P, I, PI behavior can be achieved by assigning the corresponding values for the "PControl" and "IControl" inputs. The controller parameters can be changed "on-the-fly". Thus, the values are immediately applied when the controller is switched on. The following table can be used to create the required control algorithm. If an invalid value corresponds to the figure 0, all other values are invalid.

A specific configuration is produced if all three values for "PControl" and "IControl" are set to 0 or 0s. In this case, the controller is switched off (no P-value and I-value) and the "ControlValue" output is fixed to 0. However, the

functionality for "Preset" and "Pause" remains. Thus, other values can be forced at the output using these inputs.

Controller type	PControl	IControl
Switched off	0.0	0 s
P-controller	Valid value	0 s
I-controller	0.0	Valid value
PI-controller	Valid value	Valid value

Tab. 7-21: PI-controller configuration options

The control algorithm can be displayed as the following block diagram:

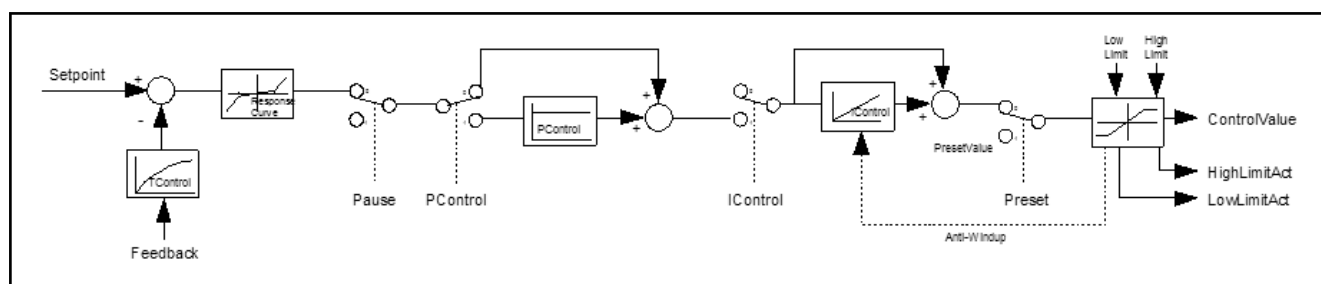


Fig. 7-14: PI-controller block diagram

Functional description

Initialization

During the first action of the function block, the variables $x_{d,k}$, $x_{d,k-1}$, $x_{d,k-2}$, y_k and y_{k-1} are initialized with the value "PresetValue". During initialization, the inputs "Setpoint" and "Feedback" have to have valid data.

If there is a rising edge at "Enable", the delay element is initialized with the value set at "Feedback", that means that "Feedback" is displayed immediately at the outputs of the delay element.

Preset value

For a trouble-free start of the controller output "ControlValue", the "Preset" input has to be set if "Enable" is activated. In this case, the internal integrators are initialized to the difference between "Setpoint" and "Feedback" scaled by "ResponseCurve". The results is displayed immediately at "ControlValue"

During this time, the inputs "Setpoint" and "Feedback" have to have valid data.

The checking of the upper/lower values is also still active, in order to ensure that the "ControlValue" is limited if the preset value ("PresetValue") is outside the limits of "HighLimit" or "LowLimit".

"PresetAck" becomes TRUE when the value is applied at "PresetValue".

"Preset" has a higher priority than the "Pause" input.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. In this case, the control deviation ("Setpoint" - scaled "Feedback") is set to 0. This causes "ControlValue" to be fixed. It can be used to keep the output constant stable.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Normal mode

If "ControlValue" exceeds or falls below the value of "HighLimit" or "LowLimit", "ControlValue" is limited to the respective value. If the controller has an integration value, an anti-reset wind-up is performed. Thus, the integral value is not added up further. In this case, the "HighLimitAct" and "LowLimitAct" outputs are set.

The controller coefficients are recalculated if controller parameters or scanning time change. Thus, the user the option can fine-tune by adjusting the controller parameters. The scanning time is measured each time the function block is called. If the scanning time deviates by more than 20% from the first one measured, it is assumed that the scanning time changed and all coefficients are recalculated. The function block should only be used in a cyclic task, as the calculations are time-consuming.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000001	"LowLimit" input is greater than the "HighLimit" input value
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000002	Controller type selection is not supported. Check inputs for "PControl" and "IControl".
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000001	The "FeedbackTControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000003	The "IControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000004	The "PControl" input is invalid.
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000001	X-value in "ResponseCurve" not in ascending order
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000002	More than one value for one x-position in "ResponseCurve"
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000003	Negative x-value in "ResponseCurve"
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-22: Generated error numbers of the IL_PIDType01 function block

7.4.7 IL_PIDType02

Brief description

The PI-controller contains an anti-reset windup integrator and a delimiter at the output. The actual value input is filtered by an adjustable PT1-element. The difference between the set value and the actual value is evaluated by a set point curve. A preset value can also be set and the controller can be paused. In this case, the controller output remains constant.

Calculation only takes place if the "Step" input changes. The last processed step is available as an output.

Interface description

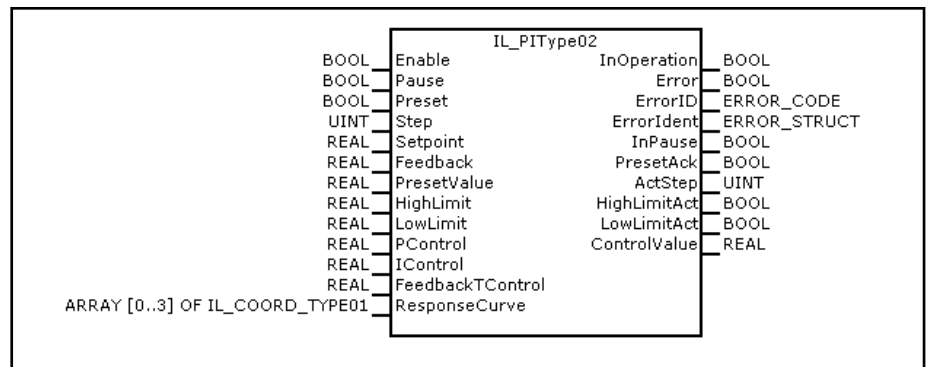


Fig. 7-15: IL_PType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "ControlValue" is either set to the preset value of "Preset-Value" (if a rising edge is set at "Preset"), or to the calculated value.
	Pause	BOOL	If TRUE, the value of "ControlValue" is not changed by the controller.
	Preset	BOOL	At a rising edge at this input, the controller output is set to the parameterizable value in "PresetValue".
	Setpoint	REAL	The desired command or reference value for the controlled process.
	Feedback	REAL	The current actual value of the process.
	PresetValue	REAL	A preset value which is displayed at "ControlValue" when there is a rising edge at "Preset".
	Step	UINT	Controller cycle step. The "ControlValue" is only calculated if "Step" changes.
	HighLimit	REAL	Upper limit for "ControlValue" If "ControlValue" exceeds this value, it is limited to this value
	LowLimit	REAL	Lower limit for "ControlValue" If "ControlValue" falls below this value, it is limited to this value
	PControl	REAL	Proportional value for the gaining value of the controller
	IControl	TIME	Integration time Tn for the integration value of the controller
	FeedbackT-Control	TIME	PT1 element filter time constant, which smoothes the "Feedback" actual value input.
	Response-Curve	ARRAY[0..3] OF IL_COORD_TYPE01	Setpoint curve for evaluating the difference between the smoothed "Feedback" actual value and the "Setpoint" value. IL_COORD_TYPE01, page 491
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the PI-controller works and the "ControlValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 354 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	PresetAck	BOOL	Value applied at "PresetValue".

	Name	Type	Comment
	ActStep	UINT	Step value to which "OutputValue" refers.
	HighLimitAct	BOOL	"ControlValue" reached upper limit
	LowLimitAct	BOOL	"ControlValue" reached lower limit
	ControlValue	REAL	Calculated controller output value

Tab. 7-23: IL_PType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Preset	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Setpoint	REAL	n.def.	n.def.	0.0	Continuous
Feedback	REAL	n.def.	n.def.	0.0	Continuous
PresetValue	REAL	n.def.	n.def.	0.0	Rising edge at Preset
HighLimit	REAL	LowLimit	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	HighLimit	0.0	Continuous
PControl	REAL	n.def.	n.def.	0.0	Continuous
IControl	REAL	0.0	n.def.	0.0	Continuous
FeedbackTControl	REAL	0.0	n.def.	0.0	Continuous
ResponseCurve	ARRAY [0..3] OF IL_CO- ORD_TYPE01			0	Rising edge at Enable

Tab. 7-24: Min./max. and default values of the IL_PType02 inputs

Controller configuration

The signal at the "Feedback" input is smoothed by a delay element of the first order. The associated filter time constant is set via "FeedbackTControl". If "FeedbackTControl" is 0s, the filter is deactivated. The difference signal from filtered "Feedback" input and "Setpoint" is scaled by a characteristic curve given in "ResponseCurve". The characteristic curve is point-symmetrical to the origin. The x-values of the characteristic curve correspond to the differential signal. The y-values correspond to the scaled difference signal. Up to four points can be specified. The sections between the points are linearly interpolated. If "ResponseCurve" is not applied, there is no scaling.

The desired P, I, PI behavior can be achieved by assigning the corresponding values for the "PControl" and "IControl" inputs. The controller parameters can be changed "on-the-fly". Thus, the values are immediately applied when the controller is switched on. The following table can be used to create the required control algorithm.

A specific configuration exists if all three values for "PControl" and "IControl" are set to 0.0. In this case, the controller is switched off (no P-value and I-value) and the "ControlValue" output is fixed to 0.0. However, the functional-

ty for "Preset" and "Pause" remains. Thus, other values can be forced at the output using these inputs.

Controller type	PControl	IControl
Switched off	0.0	0.0
P-controller	Valid value	0.0
I-controller	0.0	Valid value
PI-controller	Valid value	Valid value

Tab. 7-25: PI-controller configuration options

The control algorithm can be displayed as the following block diagram:

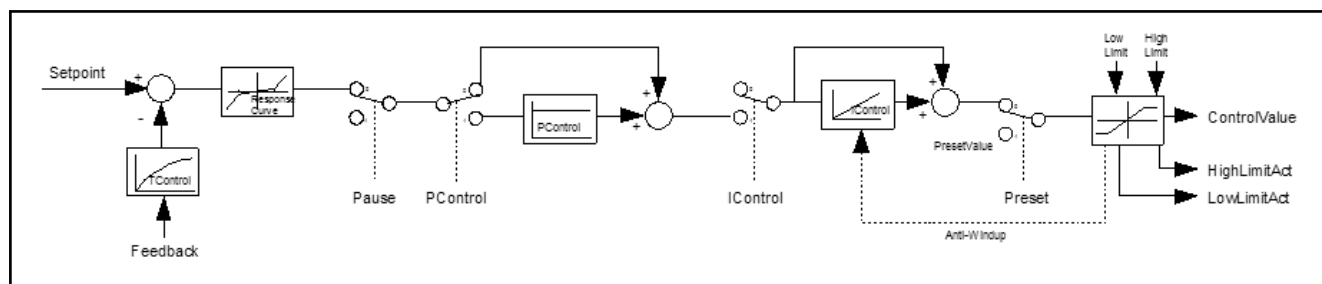


Fig. 7-16: PI-controller block diagram

Functional description

Initialization

During the first action of the function block, the variables $x_{d,k}$, $x_{d,k-1}$, $x_{d,k-2}$, y_k and y_{k-1} are initialized with the value "PresetValue". During initialization, the inputs "Setpoint" and "Feedback" have to have valid data.

If there is a rising edge at "Enable", the delay element is initialized with the value set at "Feedback", that means that "Feedback" is displayed immediately at the outputs of the delay element.

The value of the "Step" input is copied to "ActStep".

Preset value

For a trouble-free start of the controller output "ControlValue", the "Preset" input has to be set if "Enable" is activated. In this case, the internal integrators are initialized to the difference between "Setpoint" and "Feedback" scaled by "ResponseCurve". The results is displayed immediately at "ControlValue".

During this time, the inputs "Setpoint" and "Feedback" have to have valid data.

The checking of the upper/lower values is also still active, in order to ensure that the "ControlValue" is limited if the preset value ("PresetValue") is outside the limits of "HighLimit" or "LowLimit".

"PresetAck" becomes TRUE when the value is applied at "PresetValue".

"Preset" has a higher priority than the "Pause" input.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. In this case, the control deviation ("Setpoint" - scaled "Feedback") is set to 0. This causes "ControlValue" to be fixed. It can be used to keep the output constant stable.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Normal mode

The controller only calculates if the value changes at the "Step" input. The step difference between two function block calls is recorded. "IControl" and "FeedbackTControl" have steps as a time base.

If "ControlValue" exceeds or falls below the value of "HighLimit" or "LowLimit", "ControlValue" is limited to one of these values. If the controller has an integration value, an anti-reset wind-up is performed. Thus, the integral value is not added up further. In this case, the "HighLimitAct" and "LowLimitAct" outputs are set.

The controller coefficients are recalculated if the controller parameters or the step difference change. Thus, the user the option can fine-tune by adjusting the controller parameters. The step difference is measured each time the function block is called. If the step difference deviates by more than 20% from the first one measured and is not 0, then all the coefficients are recalculated.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000001	"LowLimit" input is greater than the "HighLimit" input value
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000002	Controller type selection is not supported. Check inputs for "PControl" and "IControl".
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000001	The "FeedbackTControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000003	The "IControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000004	The "PControl" input is invalid.
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000001	X-value in "ResponseCurve" not in ascending order
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000002	More than one value for one x-position in "ResponseCurve"
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000003	Negative x-value in "ResponseCurve"
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-26: Generated error numbers of the IL_PIDType02 function block

7.5 Elements (basic elements)

7.5.1 Introduction and overview

The RIL_LoopControl provides the following control-engineering basic elements:

Time-discrete	Event-controlled	Description
IL_DeadtimeType01, page 355	IL_DeadtimeType02, page 357	Dead time element (time-variable)
IL_DT1Type01, page 362	IL_DT1Type02, page 364	Differential element with first order deceleration
IL_PDT1Type01, page 367	IL_PDT1Type02, page 370	Digital filter to increase or decrease the transfer function starting from a certain frequency
IL_PT1Type01, page 373	IL_PT1Type02, page 376	First order deceleration element

Time-discrete	Event-controlled	Description
IL_TransportDelayType01 , page 378	IL_TransportDelayType02 , page 381	Dead time element (runtime-variable)
IL_DeviationResponseCurve , page 360		Evaluation graph
IL_ValueLimit , page 384		Value limitation
IL_ValueWindow , page 385		Value range check

Tab. 7-27: RIL_LoopControl filter

7.5.2 IL_DeadtimeType01

Brief description This function block implements a time-discrete dead time element with a variable dead time. The dead time can be continuously changed. The actual dead time is constantly adjusted to the preset value. The alignment to the new dead time is based on the "variable time delay" diagram.

The dead time block should be called cyclically. As it is implemented as a digital element, it can have a significant effect on the accuracy of the output. The function block measures the time between the calls (scanning time) and updates the resulting dead time if the scanning time of the preset dead time changes. The measured scanning time is also available as an output.

Interface description

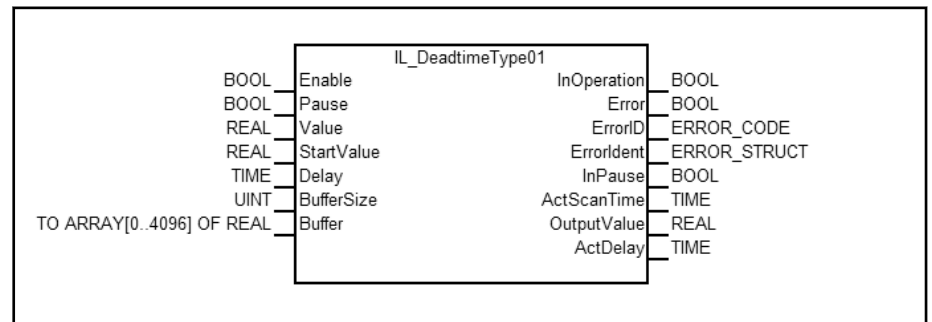


Fig. 7-17: IL_DeadtimeType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	At a rising edge, "OutputValue" is set to the "StartValue" preset value and the block is activated.
	Pause	BOOL	If TRUE, then the value of "OutputValue" is not changed.
	Value	REAL	Input value, delayed by "Delay", should be reissued.
	StartValue	REAL	A preset value displayed at the "OutputValue" at a rising edge at "Enable".
	Delay	TIME	Preset dead time
	BufferSize	UINT	Size of the buffer given in "Buffer" in elements
	Buffer	POINTER TO REAL	Buffer for buffer storing the values
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the function block is working and the "OutputValue" and "ActDelay" outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes , page 357.

	Name	Type	Comment
	InPause	BOOL	"Pause" is active and the values at "OutputValue" and "ActDelay" are frozen.
	ActScan-Time	TIME	The elapsed time since the function block was called the last time.
	OutputValue	REAL	The value delayed by "ActDelay"
	ActDelay	TIME	Current deceleration

Tab. 7-28: *IL_DeadtimeType01 function block***Min./max. and default values of the inputs**

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n.def.	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Delay	TIME	0 s	n.def.	0 s	Continuous
BufferSize	UINT	0	4096	0	Rising edge at "Enable"
Buffer	POINTER TO REAL			0	Rising edge at "Enable"

Tab. 7-29: *Min./max. and default values of the IL_DeadtimeType01 inputs***Functional description****Initialization**

At a rising edge at "Enable", after checking the "BufferSize" and "Buffer", the buffer is inputted with the value of "StartValue". "ActDelay" is initialized with 0s. "StartValue" is displayed immediately at the "OutputValue" output and remains constant until "ActDelay" equals "Delay". Only then the initial values of "Value" are displayed at the "OutputValue" output.

Normal mode

The function block is working when the "Enable" input is active. With each call, the time to the previous call is measured and issued at the "ActScan-Time" output. "ActScanTime" is also used to calculate the buffer management.

At each call, the function block stores the value at "Value" and issues this again delayed by "ActDelay" at "OutputValue". Should the "BufferSize" of the "Buffer" not be sufficient to buffer store all values for the time given in "Delay", then values will be left out. This has no effect on the resulting dead time.

The "ActDelay" output indicates the current delay of the value at "OutputValue". At a rising edge at Enable or if there is a change at "Delay", "ActDelay" is constantly updated to the value in "Delay". If "Delay" changes, while the block is active, then the dead time is constantly adjusted on the basis of the variable time delay diagram. The dead time adjustment takes a maximum of "BufferSize" * "ActScanTime".

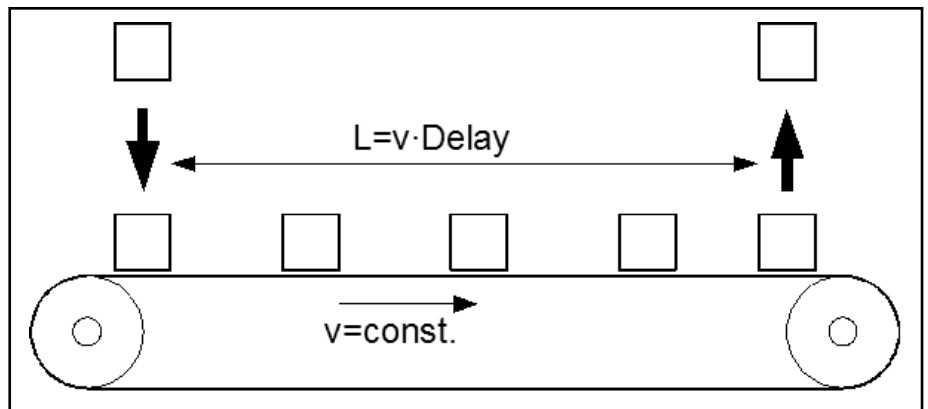


Fig. 7-18: A conveyor belt runs at a constant velocity to illustrate the dead time alignment. The packages are placed on a fixed point on the belt and removed from the belt once the dead time has elapsed. If the dead time changes, the point changes at which the packages are taken from the belt.

Pause

The "Pause" input is only evaluated if the "Enable" input is active. The function block can be stopped via "Pause". In this case, no further values will be stored and "OutputValue" and "ActDelay" remain constant.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Configuration

The size of the buffer in "Buffer" affects the accuracy of the output graph. It is possible to create a smaller buffer than 4096 values at "Buffer". Ideally the buffer will be greater than the maximum delay required, divided by the call cycle time of the function block. "BufferSize" can also be less than the delay. In this case, there is not a new value at the "OutputValue" output with each cycle. The time resolution falls.

Required buffer size:

$$\text{"BufferSize"} = 1.4 * \max(\text{"Delay"}) / \text{required resolution}$$

Adequate buffer size:

$$\text{"BufferSize"} = 1.4 * \max(\text{"Delay"}) / \text{"ActScanTime"}$$

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000001	The "Delay" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000002	The delay given in "Delay" is too large.
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000003	The "Buffer" input is not assigned
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000004	"BufferSize" is too small

Tab. 7-30: Generated error numbers of the IL_DeadtimeType01 function block

7.5.3 IL_DeadtimeType02

Brief description

This function block implements an event-controlled dead time element with a variable dead time. The dead time can be continuously changed. The actual dead time steps are constantly aligned with the preset value. The alignment to the new dead time is based on the "variable time delay" diagram.

The interval of the "Step" input should be as constant as possible. The last processed step is also available as an output.

Interface description

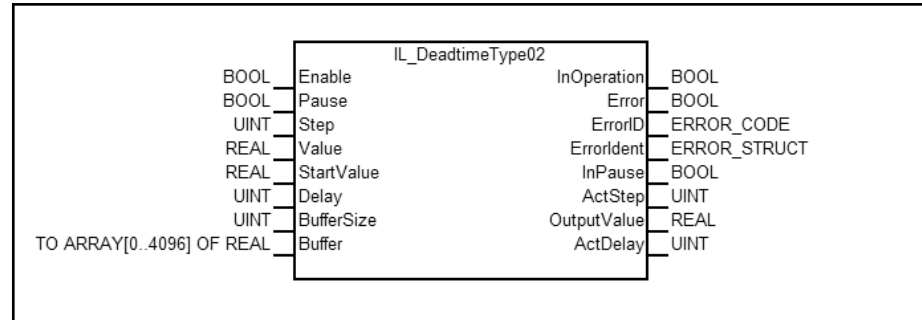


Fig. 7-19: IL_DeadtimeType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	At a rising edge, "OutputValue" is set to the "StartValue" preset value and the block is activated.
	Pause	BOOL	If TRUE, then the value of "OutputValue" is not changed.
	Step	UINT	Processing step
	Value	REAL	Input value, delayed by "Delay", should be reissued.
	StartValue	REAL	A preset value displayed at the "OutputValue" at a rising edge at "Enable".
	Delay	UINT	Number of steps to be delayed
	BufferSize	UINT	Size of the buffer given in "Buffer" in elements
	Buffer	POINTER TO REAL	Buffer for buffer storing the values
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the function block is working and the "OutputValue" and "ActDelay" outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 360 .
	InPause	BOOL	"Pause" is active and the values at "OutputValue" and "ActDelay" are frozen.
	ActStep	UINT	Last processed step.
	OutputValue	REAL	The value delayed by "ActDelay"
	ActDelay	UINT	Actual delay in steps

Tab. 7-31: IL_DeadtimeType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
Value	REAL	n.def.	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Delay	UINT	0	65335	0	Continuous
BufferSize	UINT	0	4096	0	Rising edge at "Enable"
Buffer	POINTER TO REAL			0	Rising edge at "Enable"

Tab. 7-32: Min./max. and default values of the IL_DeadtimeType02 inputs

Functional description

Initialization

At a rising edge at "Enable", after checking the "BufferSize" and "Buffer", the buffer is inputted with the value of "StartValue". "ActDelay" is initialized with 0. "StartValue" is displayed immediately at the "OutputValue" output and remains constant until "ActDelay" equals "Delay". Only then the initial values of "Value" are displayed at the "OutputValue" output.

Normal mode

The function block is working when the "Enable" input is active. With each call, "ActStep" is updated to "Step". Processing only takes place if the "Step" input changes. The difference between the steps is also used for buffer management.

At each call, the function block stores the value at "Value" and reissues it, delayed by "ActDelay" steps, at "OutputValue". Should the "BufferSize" of the "Buffer" not be sufficient to buffer store all values for the number of steps given in "Delay", then values will be left out. This has no effect on the resulting dead time.

The "ActDelay" output indicates the current delay of the value at "OutputValue". At a rising edge at Enable or if there is a change at "Delay", "ActDelay" is constantly updated to the value in "Delay". If "Delay" changes, while the block is active, then the dead time is constantly adjusted on the basis of the variable time delay diagram. The dead time alignment takes a maximum of "BufferSize" steps.

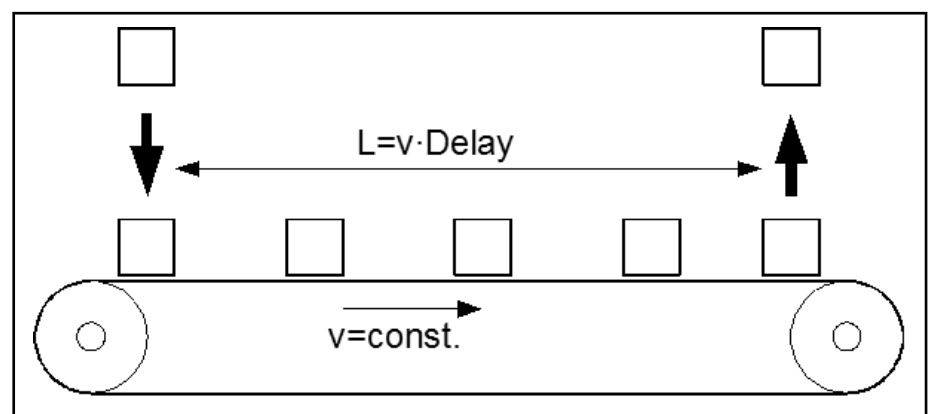


Fig. 7-20:

A conveyor belt runs at a constant velocity to illustrate the dead time alignment. The packages are placed on a fixed point on the belt and removed from the belt once the dead time has elapsed. If the dead time changes, the point changes at which the packages are taken from the belt.

Pause

The "Pause" input is only evaluated if the "Enable" input is active. The function block can be stopped via "Pause". In this case, no further values will be stored and "OutputValue" and "ActDelay" remain constant.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated

Configuration The size of the buffer in "Buffer" affects the accuracy of the output graph. It is possible to create a smaller buffer than 4096 values at "Buffer". Ideally the buffer is larger than the maximum delay required in steps. "BufferSize" can also be less than the delay. In this case, there is not a new value at the "OutputValue" output with each cycle. The resolution, based on the steps, falls.

Required buffer size:

$\text{"BufferSize"} = 1.4 * \max(\text{"Delay"}) / \text{required resolution}$

Adequate buffer size:

$\text{"BufferSize"} = 1.4 * \max(\text{"Delay"})$

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000001	The "Delay" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000002	The delay given in "Delay" is too large.
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000003	The "Buffer" input is not assigned
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000004	"BufferSize" is too small or too large.

Tab. 7-33: Generated error numbers of the IL_DeadtimeType02 function block

7.5.4 IL_DeviationResponseCurve

Brief description This function block converts an input signal point-symmetrically to the origin on a command value curve. The set point curve is given by up to 4 points. It is linearly interpolated between the points. The first section goes from the origin to the first point. From the last point onwards to infinity there is an increase of 1.0. The X-coordinates of the points have to increase in a positive direction and may not be smaller than 0.

Interface description

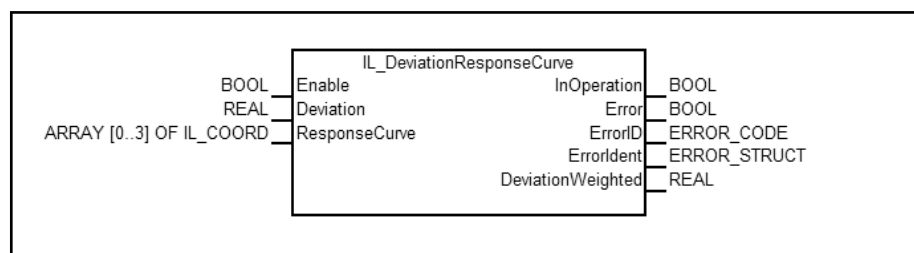


Fig. 7-21: IL_DeviationResponseCurve function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, the block is active.
	Deviation	REAL	Input value, which can be evaluated by the curve given in "ResponseCurve".

	Name	Type	Comment
	Response-Curve	ARRAY[0..3] OF IL_COORD_TYPE01	Setpoint curve consisting of value pairs IL_COORD_TYPE01, page 491
VAR_OUTPUT	InOperation	BOOL	The function block has completed initialization and "Deviation-Weighted" is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error division, see Error codes, page 362
	Deviation-Weighted	REAL	Deviation scaled by "ResponseCurve".

Tab. 7-34: IL_DeviationResponseCurve function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Deviation	REAL	n.def.	n.def.	0.0	Continuous
ResponseCurve	ARRAY [0..3] OF IL_COORD_TYPE01				Rising edge at "Enable"

Tab. 7-35: Min./max. and default values of the IL_DeviationResponseCurve inputs

Functional description**Initialization**

At a rising edge at "Enable", an examination of "ResponseCurve" is carried out and following successful examination, "InOperation" is set. The value in "DeviationWeighted" is then valid.

Normal mode

The function block is working when the "Enable" input is active. With each call, the value in "Deviation" is transferred by means of the set point curve given in "ResponseCurve" and is issued at "DeviationWeighted".

Configuration

The characteristic curve defined in "ResponseCurve" consists of four points. Negative deviations are processed by point reflecting the curve at the origin. This results in a total of nine linear sections. The section s5 runs through the origin. The sections s1 and s9 have a slope of 1 and each run into positive or negative infinity.

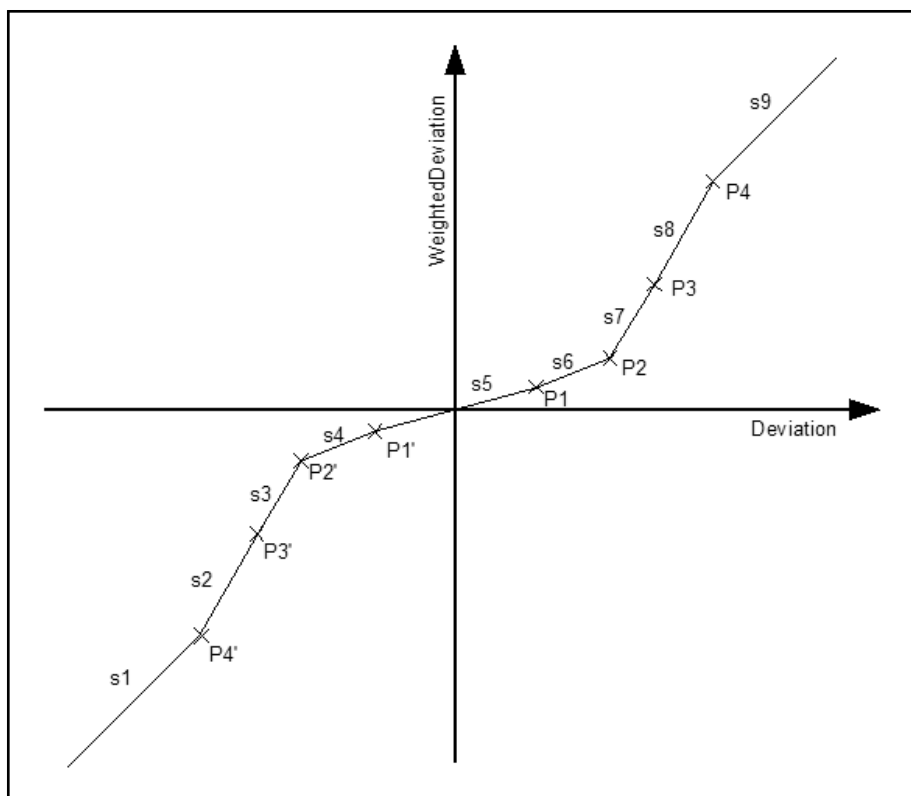


Fig. 7-22: Setpoint curve

Points P1 to P4 are arranged in an array made up of IL_COORD_TYPE01. If fewer than four points are required, the X-coordinates of the points not required (beginning with the last point) are set to 0. The X-values have to be positive and rise in a positive direction. It is permitted to use the same point twice.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000001	X-value in "ResponseCurve" not in ascending order
INPUT_INVALID_ERROR, 16#0001	16#00000C05	16#00000002	More than one value for one x-position in "ResponseCurve"
INPUT_RANGE_ERROR, 16#0006	16#00000C05	16#00000003	Negative x-value in "ResponseCurve"

Tab. 7-36: Generated error numbers of the IL_DeviationResponseCurve function block

7.5.5 IL_DT1Type01

Brief description This function block implements a time-discrete differential element with first order deceleration. The calculation takes place continuously, that is to say each time the function block is called. The function block should be called cyclically, in order to ensure as consistent a calculation as possible. The cycle time measured by the function block is also available as an output.

Interface description

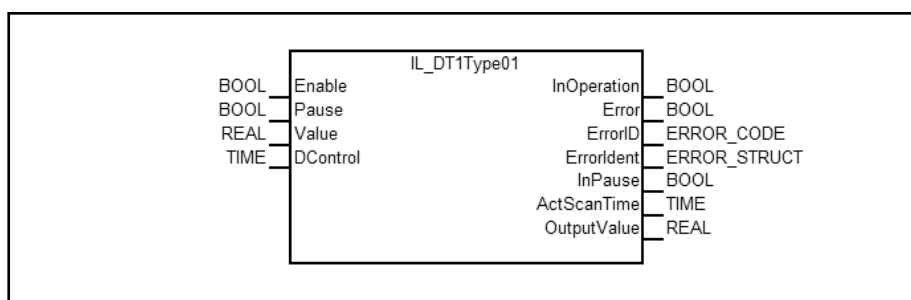


Fig. 7-23: IL_DT1Type01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the preset value of "StartValue" or to the calculated value.
	Pause	BOOL	If TRUE, then the value of "OutputValue" is not changed by the controller.
	Value	REAL	The current actual value of the process.
	DControl	TIME	Time constant Tn of the DT1 element.
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the DT1 element is working and the "OutputValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 364 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	ActScan-Time	TIME	The elapsed time since the function block was called the last time.
	OutputValue	REAL	Calculated controller output value

Tab. 7-37: IL_DT1Type01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
DControl	TIME	0 s	n.def.	1s	Continuous

Tab. 7-38: Min./max. and default values of the IL_DT1Type01 inputs

Functional description

The function block issues the calculated value of the implemented DT1 function at the "OutputValue" output. The calculation is carried out continuously when "Enable" is set. The "Value" input value is only transferred at a rising edge at the "Enable" input and is pending for the internal calculation. All inputs are continuously processed.

During the block's initialization process, the "Value" input variables, as well as the "DControl" time constant, are transferred. "OutputValue" is initialized with 0.0.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If a TRUE is pending at "Pause", no calculation takes place and the "OutputValue" value is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Normal mode

The step response of a DT1 element has the line of a curve approximating 0.

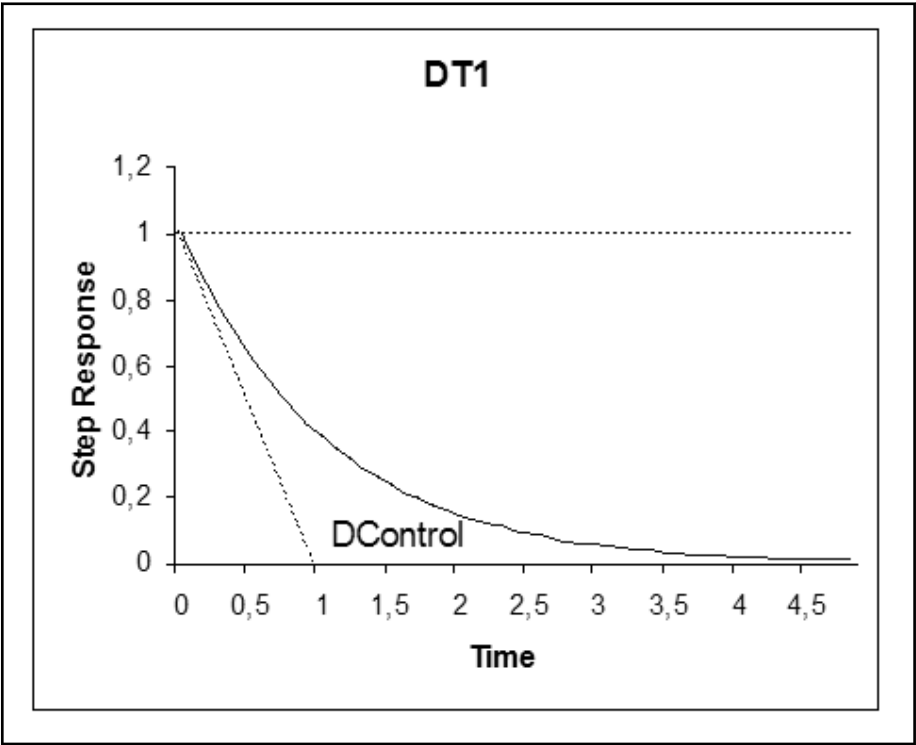


Fig. 7-24: IL_DT1Type01 step response

The transfer function is as follows:

$$G(s) = \frac{s}{1 + s \cdot DControl}$$

"DControl" indicates the delay time constant. If "DControl" is 0.0, then the DT1 element is deactivated and the "OutputValue" output is set to 0.0.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR,16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-39: Generated error numbers of the IL_DT1Type01 function block

7.5.6 IL_DT1Type02

Brief description This function block implements an event-controlled differential element with first order deceleration. The calculation is event-controlled, that is to say at every change at the Step input. The last processed step is available at the output.

Interface description

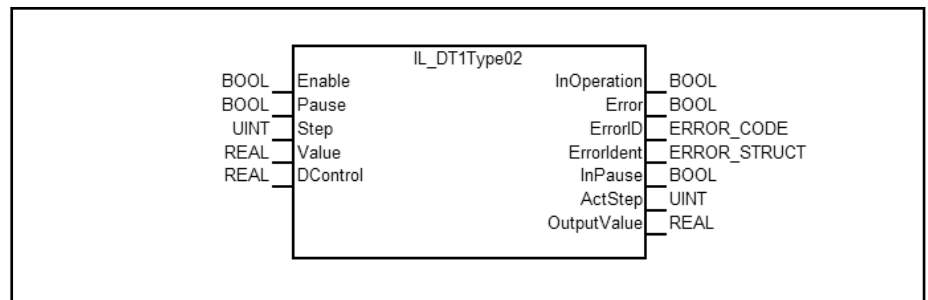


Fig. 7-25: IL_DT1Type02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the preset value of "StartValue" or to the calculated value.
	Pause	BOOL	If TRUE, then the value of "OutputValue" is not changed by the controller.
	Step	UINT	Processing step
	Value	REAL	The current actual value of the process.
	DControl	REAL	Time constant Tn of the DT1 element.
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the DT1 element is working and the "OutputValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 366 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	ActStep	UINT	Last processed step
	OutputValue	REAL	Calculated controller output value

Tab. 7-40: IL_DT1Type02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
DControl	REAL	0.0	n.def.	1.0	Continuous

Tab. 7-41: Min./max. and default values of the IL_DT1Type02 inputs

Functional description

The function block issues the calculated value of the implemented DT1 function at the "OutputValue" output. The calculation takes place when there is a change at the "Step" input when "Enable" is set. The interval at the "Step" input is processed, that is to say more than one step is calculated if required. The "Value" input value is only transferred at a rising edge at the "Enable"

input and is pending for the internal calculation. All inputs are continuously processed.

During the block's initialization process, "Value" input variables, as well as the "DControl" time constant are transferred. "OutputValue" is initialized with 0.0.

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If a TRUE is pending at "Pause", no calculation takes place and the "OutputValue" value is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Normal mode

The step response of a DT1 element has the line of a curve approximating 0.

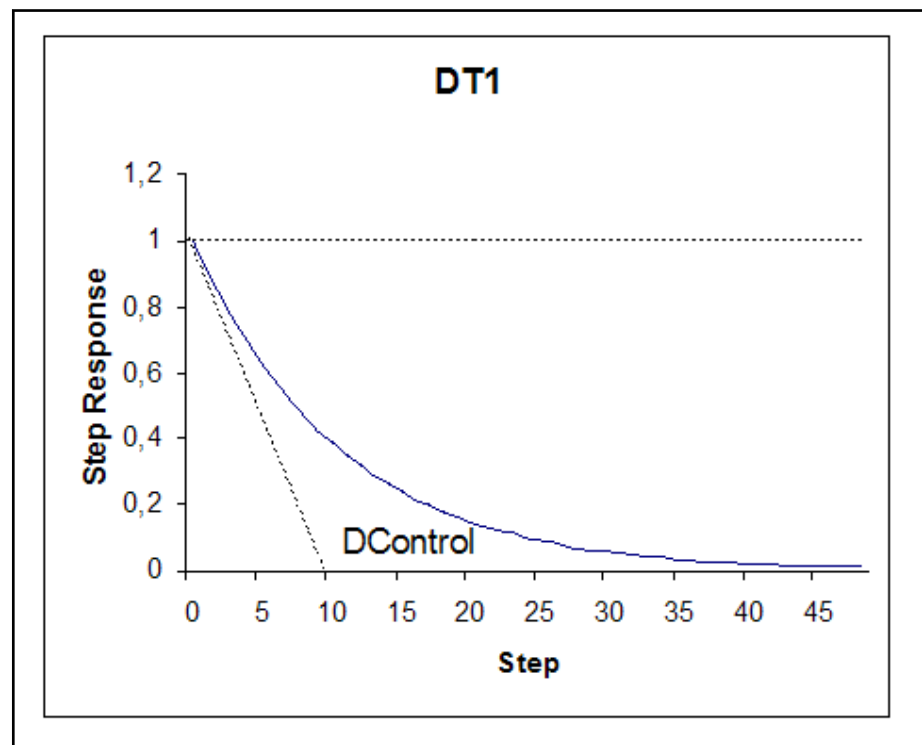


Fig. 7-26: IL_DT1Type02 step response

The transfer function is as follows:

$$G(s) = \frac{s}{1 + s \cdot DControl}$$

"DControl" indicates the delay time constant. If "DControl" is 0.0, then the DT1 element is deactivated and the "OutputValue" output is set to 0.0 each time there is a change at the "Step" input.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR,16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000002	The "DControl" input is invalid.

Tab. 7-42: Generated error numbers of the IL_DT1Type02 function block

7.5.7 IL_PDT1Type01

Brief description The IL_PDT1Type01 function block implements a time-discrete PDT1 filter. The digital filter can be used to increase or decrease the transfer function starting from a specific frequency.

The calculation takes place continuously, that is to say each time the function block is called. The function block should be called cyclically, in order to ensure as consistent a calculation as possible. The cycle time measured by the function block is also available as an output.

Interface description

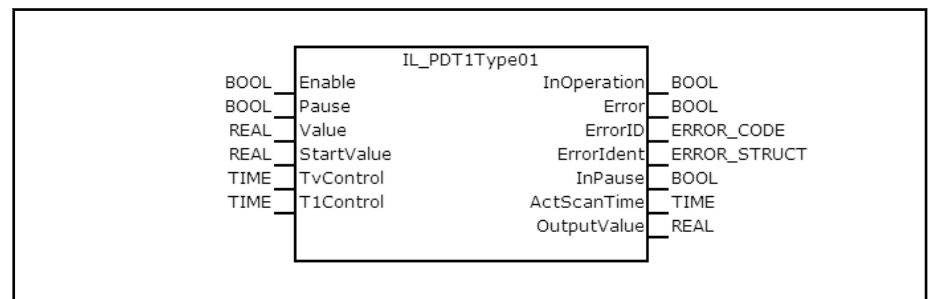


Fig. 7-27: IL_PDT1Type01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	TvControl	TIME	Derivative time constant
	T1Control	TIME	Time constant
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PT1 element is working and the "OutputValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 370 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	ActScanTime	TIME	The time elapsed since the latest function block call.
	OutputValue	REAL	The calculated filter value

Tab. 7-43: IL_PDT1Type01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
TvControl	TIME	0 s	n.def.	1ms	Continuous
T1Control	TIME	0 s	n.def.	1ms	Continuous

Tab. 7-44: Min./max. and default values of the IL_PDT1Type01 inputs

Functional description

The IL_PDT1Type01 function block implements a time-discrete PDT1 filter. The digital filter can be used to increase or decrease the transfer function starting from a specific frequency. The function block operates time-discretely.

The response depends on the relation between the two time constants:

For $T_v > T_1$: The signal is attenuated above T_v and T_1 .

For $T_v < T_1$: The signal is amplified above T_v and T_1 .

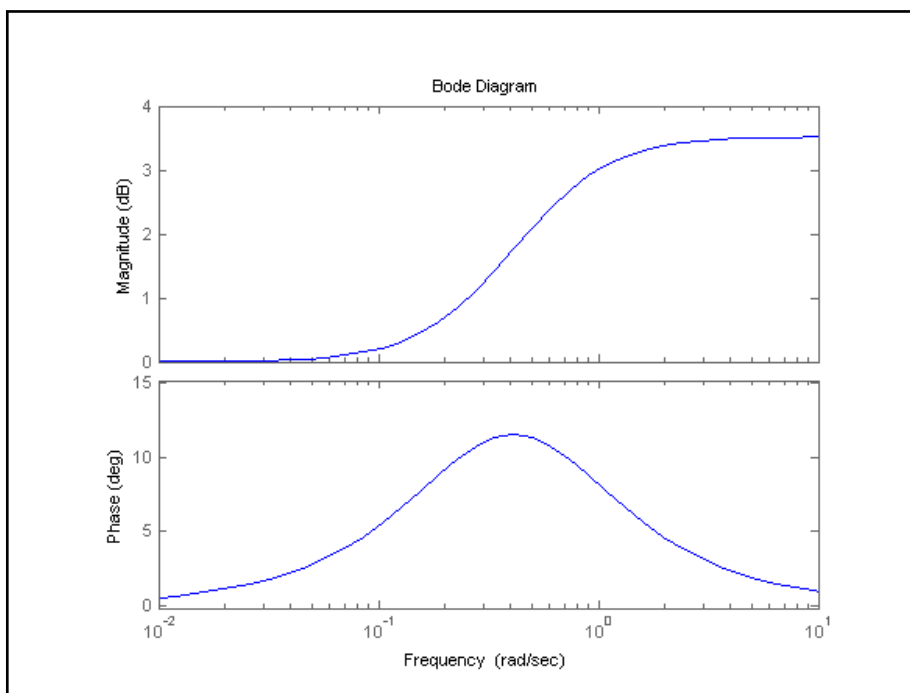


Fig. 7-28: Bode chart for $T_v > T_1$

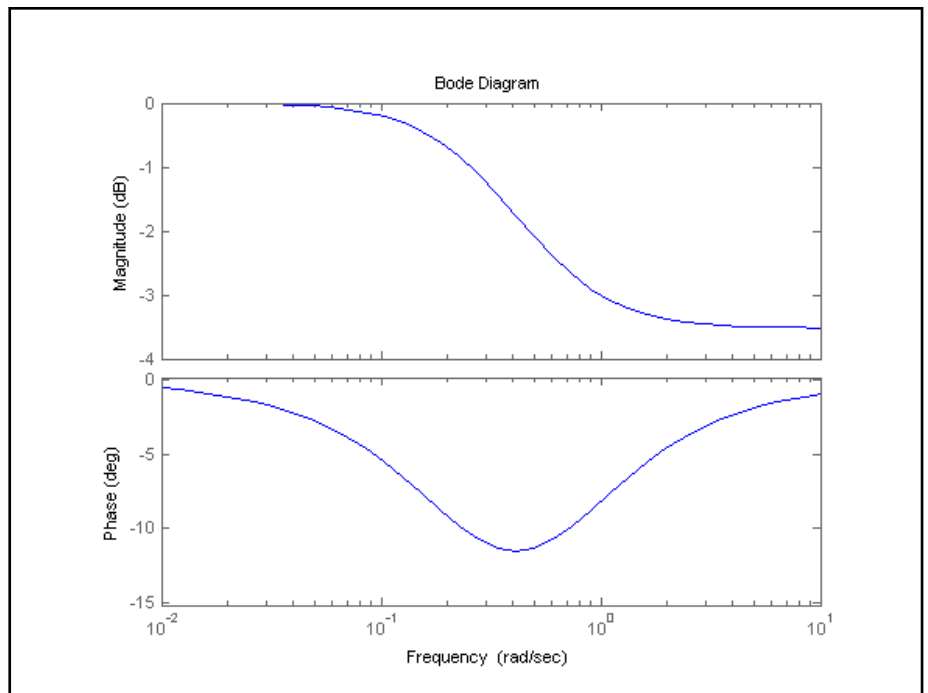


Fig. 7-29: Bode chart for $T_v < T_1$

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. The result is output at the "OutputValue" output.

T_v Control, T_1 Control

These two inputs determine the filter response. " T_v Control" is the derivative time. The ratio of T_v/T_1 is determined by the steepness of the dropping.

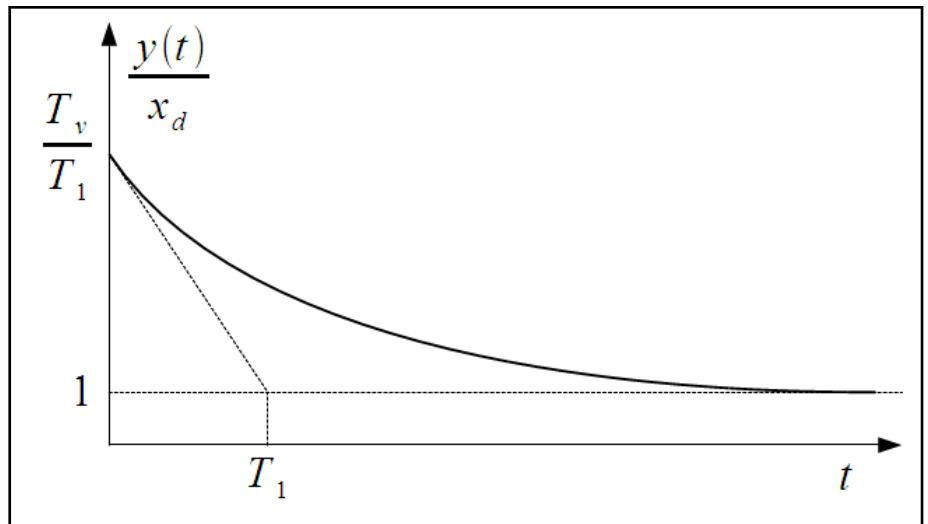


Fig. 7-30: Jump response of a PDT1 element

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Time response The IL_PDT1Type01 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "StartValue" input. At each call, the function block reads a value from the inputs "Value", "TvControl" and "T1Control" and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000006	The "TvControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000007	The "T1Control" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-45: Generated error numbers of the IL_PDT1Type01 function block

7.5.8 IL_PDT1Type02

Brief description The IL_PDT1Type02 function block implements a event-controlled PDT1 filter. The digital filter can be used to increase or decrease the transfer function starting from a specific frequency.

The calculation is event-controlled, that is to say at every change at the Step input. The last processed step is available at the output.

Interface description

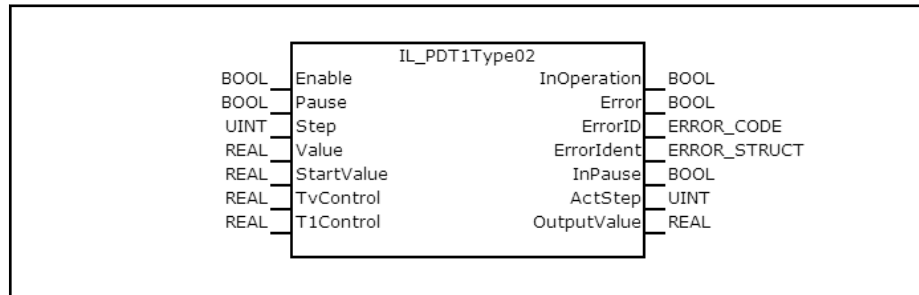


Fig. 7-31: IL_PDT1Type02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Step	UINT	Processing step
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	TvControl	TIME	Derivative time constant
	T1Control	TIME	Time constant
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PT1 element is working and the "OutputValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.

	Name	Type	Comment
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 373 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	ActStep	UINT	Step to which "OutputValue" refers.
	OutputValue	REAL	The calculated filter value

Tab. 7-46: IL_PDT1Type02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Value	REAL	n. def.	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
TvControl	TIME	>0s	n.def.	1.0	Continuous
T1Control	TIME	>0s	n.def.	1.0	Continuous

Tab. 7-47: Min./max. and default values of the IL_PDT1Type02 inputs

Functional description

The IL_PDT1Type02 function block implements a event-controlled PDT1 filter. The digital filter can be used to increase or decrease the transfer function starting from a specific frequency. The function block operates event-controlled. That means that a new value is processed when the "Step" input is changed.

The response depends on the relation between the two time constants:

For $T_v > T_1$: The signal is attenuated above T_v and T_1 .

For $T_v < T_1$: The signal is amplified above T_v and T_1 .

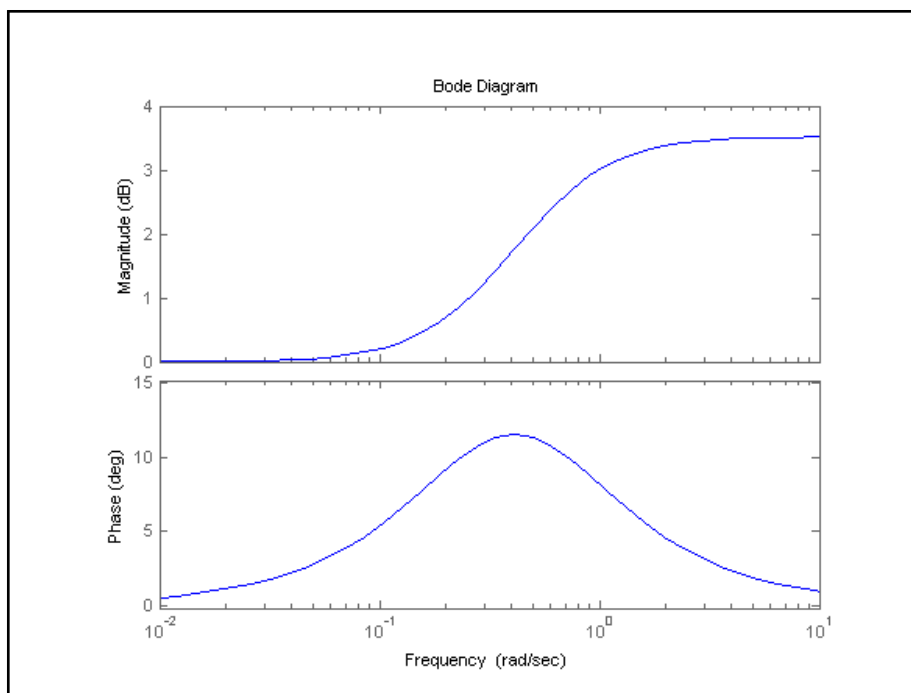


Fig. 7-32: Bode chart for $T_v > T_1$

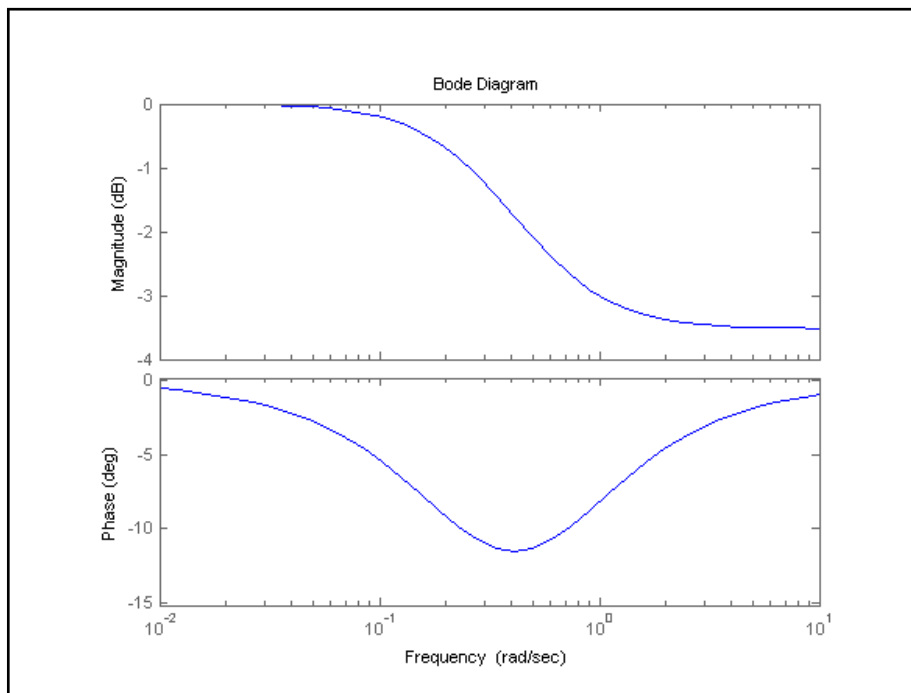


Fig. 7-33: Bode chart for $T_v < T_1$

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. The result is output at the "OutputValue" output.

TvControl, T1Control

These two inputs determine the filter response. "TvControl" is the derivative time. The ratio of T_v/T_1 is determined by the steepness of the dropping.

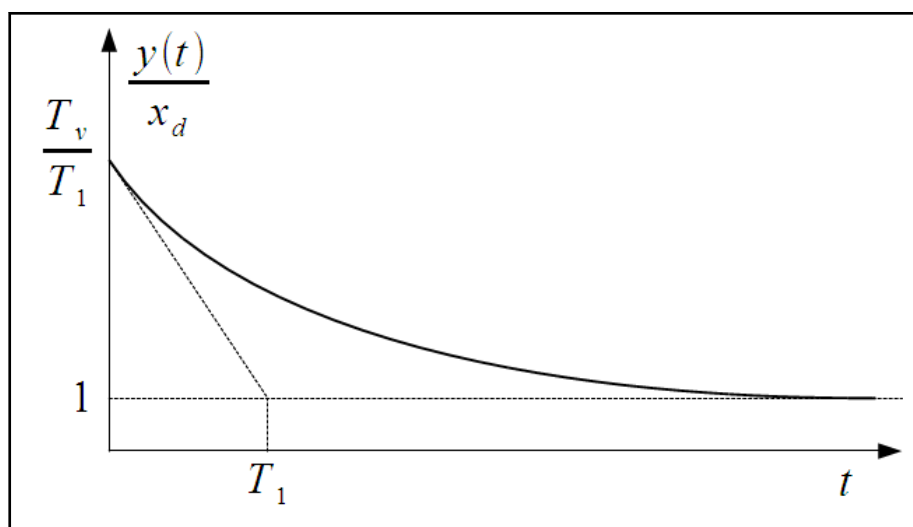


Fig. 7-34: Jump response of a PDT1 element

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Time response

The IL_PDT1Type02 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "StartValue" input. At each call, the function block reads a value from the inputs "Value", "TvControl" and "T1Control" and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000006	The "TvControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000007	The "T1Control" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-48: Generated error numbers of the IL_PDT1Type02 function block

7.5.9 IL_PT1Type01

Brief description

This function block implements a time-discrete first order delay element. The calculation takes place continuously, that is to say each time the function block is called. The function block should be called cyclically, in order to ensure as consistent a calculation as possible. The cycle time measured by the function block is also available as an output.

Interface description

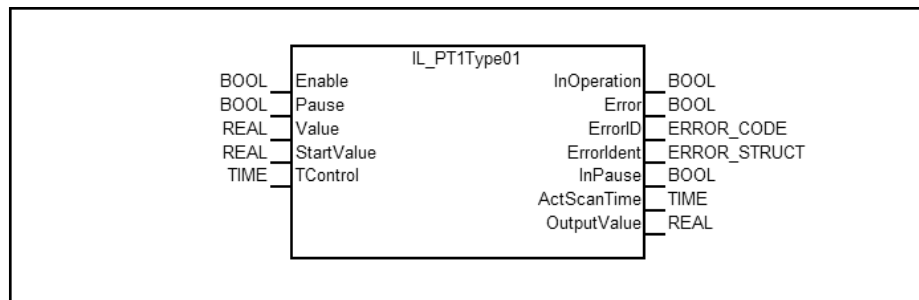


Fig. 7-35: IL_PT1Type01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the preset value of "StartValue" or to the calculated value.
	Pause	BOOL	If TRUE, then the value at "OutputValue" remains constant.
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	TControl	TIME	Time constant Tn of the PT1 element.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the PT1 element is working and the "OutputValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 375 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	ActScan-Time	TIME	The time elapsed since the latest function block call.
	OutputValue	REAL	The calculated output value

Tab. 7-49: IL_PT1Type01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
TControl	TIME	0 s	n.def.	1s	Continuous

Tab. 7-50: Min./max. and default values of the IL_PT1Type01 inputs

Functional description

The function block issues the calculated value of the first order delay function at the "OutputValue" output. The calculation is carried out continuously when the "Enable" input is set. The "StartValue" input values are only transferred at a rising edge at the "Enable" input. It is then pending at the "OutputValue"

output and is used as initial value for the internal integrator. All other inputs are continuously processed.

Start value

An initial value for the function can optionally be transferred to the block via the "StartValue" input variable. During the block's initialization process, this value is transferred along with the other input variables, such as "Value" and the time constant "TControl".

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If a TRUE is pending at "Pause", no calculation takes place and the "OutputValue" value is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output is updated, while "Pause" is active.

Normal mode

The step response of a PT1 element has the line of a curve similar to the "Value" command value.

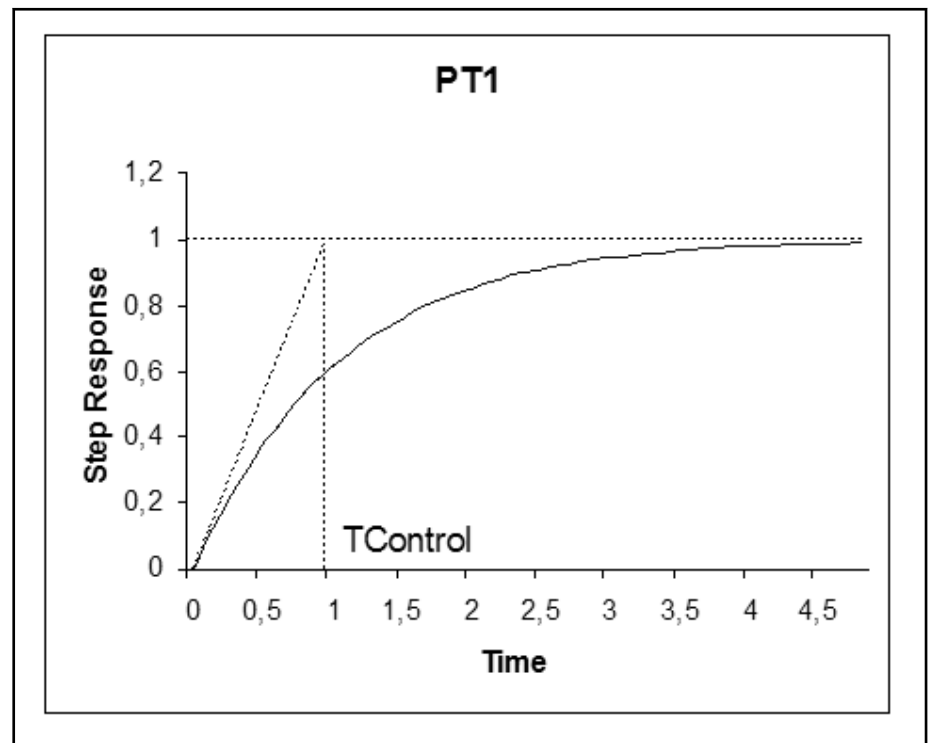


Fig. 7-36: IL_PT1Type01 step response

The transfer function is as follows:

$$G(s) = \frac{1}{1 + s \cdot TControl}$$

"TControl" indicates the delay time constant. If "TControl" is 0s, then the PT1 element is deactivated and the "Value" input is copied directly to "OutputValue".

The "TControl" and "Value" inputs are transferred each time the function block is called, while "Enable" is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR,16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.
INPUT_RANGE_ERROR	16#00000C02	16#00000001	The "TControl" input is invalid.

Tab. 7-51: Generated error numbers of the IL_PT1Type01 function block

7.5.10 IL_PT1Type02

Brief description This function block implements an event-controlled first order delay element. The calculation is event-controlled, that is to say at every change at the Step input. The last processed step is available at the output.

Interface description

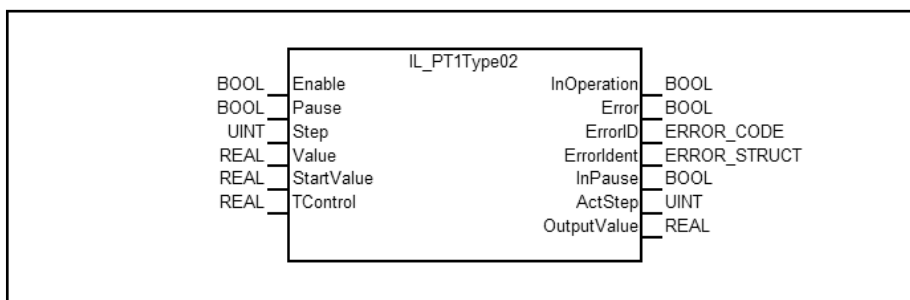


Fig. 7-37: IL_PT1Type02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the preset value of "StartValue" or to the calculated value.
	Pause	BOOL	If TRUE, then the value at "OutputValue" remains constant.
	Step	UINT	Processing step
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	TControl	REAL	Time constant Tn of the PT1 element.
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the PT1 element is working and the "OutputValue" controller output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 378 .
	InPause	BOOL	"Pause" is active and the value at "ControlValue" is frozen.
	ActStep	UINT	Step to which "OutputValue" refers.
	OutputValue	REAL	Calculated controller output value

Tab. 7-52: IL_PT1Type02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
TControl	REAL	0.0	n.def.	1.0	Continuous

Tab. 7-53: Min./max. and default values of the IL_PT1Type02 inputs

Functional description

The function block issues the calculated value of the implemented PT1 function at the "OutputValue" output. The calculation takes place when there is a change at the "Step" input when "Enable" is set. The interval at the "Step" input is processed, that is to say more than one step is calculated if required. The "StartValue" input value is only transferred at a rising edge at the "Enable" input. It is then pending at the "OutputValue" output and is used as initial value for the internal integrator. All other inputs are continuously processed.

Start value

An initial value for the function can optionally be transferred to the block via the "StartValue" input variable. During the block's initialization process, this value is transferred along with the other input variables, such as "Value" and the time constant "TControl".

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If a TRUE is pending at "Pause", no calculation takes place and the "OutputValue" value is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Normal mode

The step response of a PT1 element has the line of a curve similar to the "Value" command value.

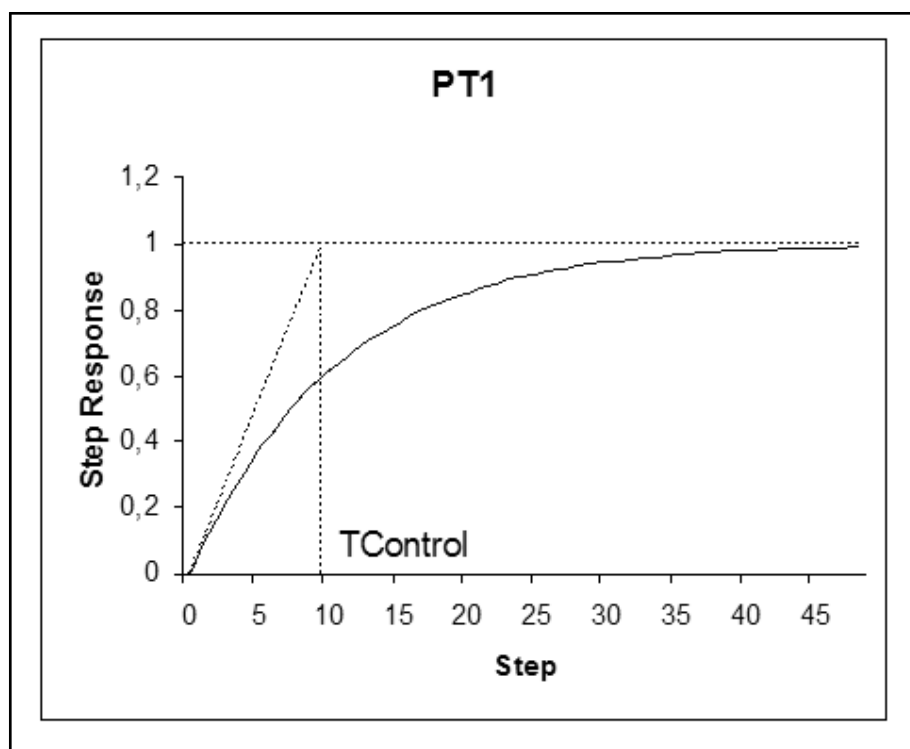


Fig. 7-38: *IL_PT1Type02 step response*

The transfer function is as follows:

$$G(s) = \frac{1}{1 + s \cdot TControl}$$

"TControl" indicates the delay time constant. If "TControl" is 0.0, then the PT1 element is deactivated and the "Value" input is copied directly to "OutputValue" each time there is a change at the "Step" input.

The "TControl" and "Value" inputs are transferred each time the function block is called, while "Enable" is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.
INPUT_RANGE_ERROR. 16#0006	16#00000C02	16#00000001	The "TControl" input is invalid.

Tab. 7-54: *Generated error numbers of the IL_PT1Type02 function block*

7.5.11 IL_TransportDelayType01

Brief description

This function block implements a time-discrete dead time element with a variable dead time. The dead time can be continuously changed. The actual dead time is constantly adjusted to the preset value. The adjustment to the new dead time is based on the "variable transport delay" diagram.

The dead time block should be called cyclically. As it is implemented as a digital element, it can have a significant effect on the accuracy of the output. The function block measures the time between the calls (scanning time) and updates the resulting dead time if the scanning time changes substantially or

if the preset dead time changes. The measured scanning time is also available as an output.

Interface description

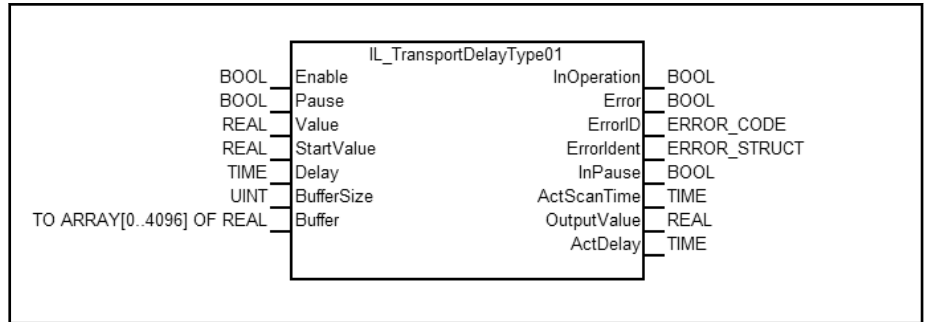


Fig. 7-39: IL_TransportDelayType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	At a rising edge, "OutputValue" is set to the "StartValue" preset value. If TRUE, the block is active.
	Pause	BOOL	If TRUE, then the value of "OutputValue" is not changed.
	Value	REAL	Input value, delayed by "Delay", should be reissued.
	StartValue	REAL	A preset value displayed at the "OutputValue" at a rising edge at "Enable".
	Delay	TIME	Preset dead time
	BufferSize	UINT	Size of the buffer given in "Buffer" in elements
	Buffer	POINTER TO REAL	Buffer for buffer storing the values
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the function block is working and the "OutputValue" and "ActDelay" outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 381 .
	InPause	BOOL	"Pause" is active and the values at "OutputValue" and "ActDelay" are frozen.
	ActScanTime	TIME	The elapsed time since the function block was called the last time.
	OutputValue	REAL	The value delayed by "ActDelay"
	ActDelay	TIME	Actual delay

Tab. 7-55: IL_TransportDelayType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n.def.	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"

Name	Type	Min. value	Max. value	Default value	Effective
Delay	TIME	0 s	n.def.	0 s	Continuous
BufferSize	UINT	0	4096	0	Rising edge at "Enable"
Buffer	POINTER TO REAL			0	Rising edge at "Enable"

Tab. 7-56: Min./max. and default values of the IL_TransportDelayType01 inputs

Functional description

Initialization

At a rising edge at "Enable", after checking the "BufferSize" and "Buffer", the buffer is inputted with the value of "StartValue". "ActDelay" is initialized with "Delay". "StartValue" is displayed immediately at the "OutputValue" output and remains there until the period of time in "Delay" has elapsed. Only then the initial values of "Value" are displayed at the "OutputValue" output.

Normal mode

The function block is working when the "Enable" input is active. With each call, the time to the previous call is measured and issued at the "ActScanTime" output. "ActScanTime" is also used to calculate the buffer management.

At each call, the function block stores the value at "Value" and issues this again delayed by "ActDelay" at "OutputValue". Should the "BufferSize" of the "Buffer" not be sufficient to buffer store all values for the time given in "Delay", then values will be left out. This has no effect on the resulting dead time.

The "ActDelay" output indicates the current delay of the value at "OutputValue". If "Delay" changes, "ActDelay" is continuously updated to the value in "Delay". If "Delay" changes, while the block is active, then the dead time is constantly adjusted on the basis of the variable transport delay diagram. The dead time adjustment takes a maximum of "BufferSize" * "ActScanTime".

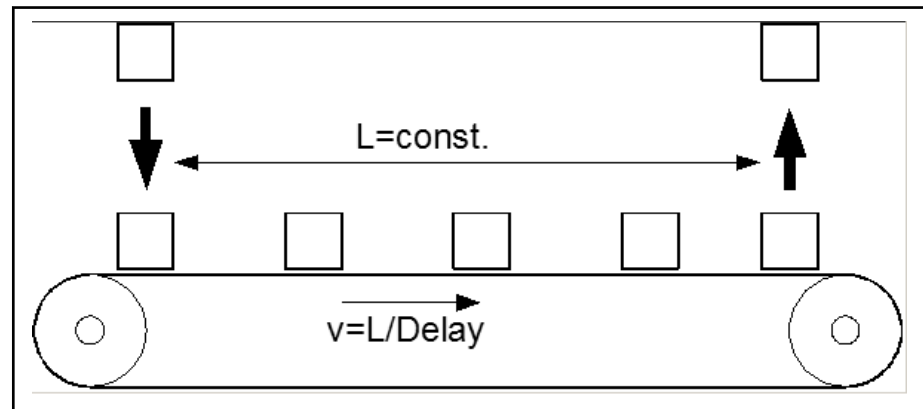


Fig. 7-40:

A conveyor belt runs at a variable velocity to illustrate the dead time alignment. The packages are placed on a fixed point on the belt and removed from the belt once a fixed period has elapsed. The velocity of the belt is set in such a way that every package spends the length of the dead time on the belt. Should the dead time change, then the belt velocity is adjusted accordingly.

Pause

The "Pause" input is only evaluated if the "Enable" input is active. The function block can be stopped via "Pause". In this case, no further values will be stored and "OutputValue" and "ActDelay" remain constant.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Configuration The size of the buffer in "Buffer" affects the accuracy of the output graph. It is possible to create a smaller buffer than 4096 values at "Buffer". Ideally the buffer will be greater than the maximum delay required, divided by the call cycle time of the function block. "BufferSize" can also be less than the delay. In this case, there is not a new value at the "OutputValue" output with each cycle. The time resolution falls.

Required buffer size:

"BufferSize" = max("Delay")/required resolution

Adequate buffer size:

"BufferSize" = max("Delay")/"ActScanTime"

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000001	The "Delay" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000002	The delay given in "Delay" is too large.
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000003	The "Buffer" input is not assigned
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000004	"BufferSize" is too small

Tab. 7-57: Generated error numbers of the IL_TransportDelayType01 function block

7.5.12 IL_TransportDelayType02

Brief description This function block implements an event-controlled dead time element with a variable dead time. The dead time can be continuously changed. The actual dead time is constantly adjusted to the preset value. The adjustment to the new dead time is based on the "variable transport delay" diagram.

The dead time block should be called cyclically. The interval of the "Step" input should be as constant as possible. The last processed step is also available as an output.

Interface description

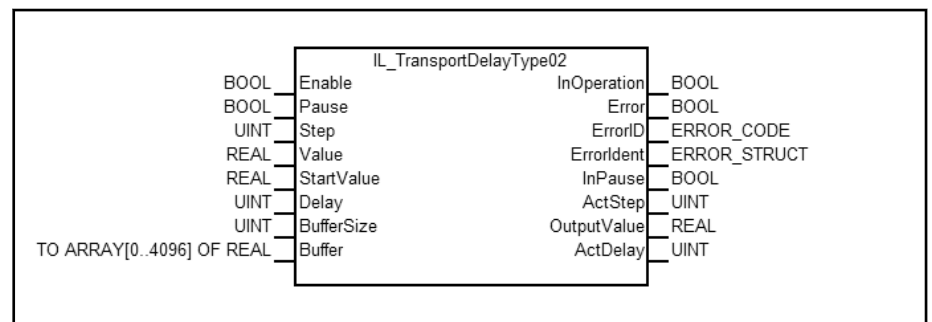


Fig. 7-41: IL_TransportDelayType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	At a rising edge, "OutputValue" is set to the "StartValue" preset value. If TRUE, the block is active.
	Pause	BOOL	If TRUE, then the value of "OutputValue" is not changed.
	Step	UINT	Processing step
	Value	REAL	Input value, delayed by "Delay", should be reissued.

	Name	Type	Comment
	StartValue	REAL	A preset value displayed at the "OutputValue" at a rising edge at "Enable".
	Delay	UINT	Number of steps to be delayed
	BufferSize	UINT	Size of the buffer given in "Buffer" in elements
	Buffer	POINTER TO REAL	Buffer for buffer storing the values
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the function block is working and the "OutputValue" and "ActDelay" outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 383 .
	InPause	BOOL	"Pause" is active and the values at "OutputValue" and "ActDelay" are frozen.
	ActStep	UINT	Last processed step.
	OutputValue	REAL	The value delayed by "ActDelay"
	ActDelay	UINT	Actual delay in steps

Tab. 7-58: IL_TransportDelayType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	65335	0	Continuous
Value	REAL	n.def.	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Delay	UINT	0	65335	0	Continuous
BufferSize	UINT	0	4096	0	Rising edge at "Enable"
Buffer	POINTER TO REAL			0	Rising edge at "Enable"

Tab. 7-59: Min./max. and default values of the IL_TransportDelayType02 inputs

Functional description Initialization

At a rising edge at "Enable", after checking the "BufferSize" and "Buffer", the buffer is inputted with the value of "StartValue". "ActDelay" is initialized with "Delay". "StartValue" is displayed immediately at the "OutputValue" output and remains constant until the number of steps given in "Delay" has passed. Only then the initial values of "Value" are displayed at the "OutputValue" output.

Normal mode

The function block is working when the "Enable" input is active. With each call, "ActStep" is updated to "Step". Processing only takes place if the "Step"

input changes. The difference between the steps is also used for buffer management.

At each call, the function block stores the value at "Value" and reissues it, delayed by "ActDelay" steps, at "OutputValue". Should the "BufferSize" of the "Buffer" not be sufficient to buffer store all values for the number of steps given in "Delay", then values will be left out. This has no effect on the resulting dead time.

The "ActDelay" output indicates the current delay of the value at "OutputValue". If "Delay" changes, "ActDelay" is continuously updated to the value in "Delay." If "Delay" changes, while the block is active, then the dead time is constantly adjusted on the basis of the variable transport delay diagram. The dead time alignment takes a maximum of "BufferSize" steps.

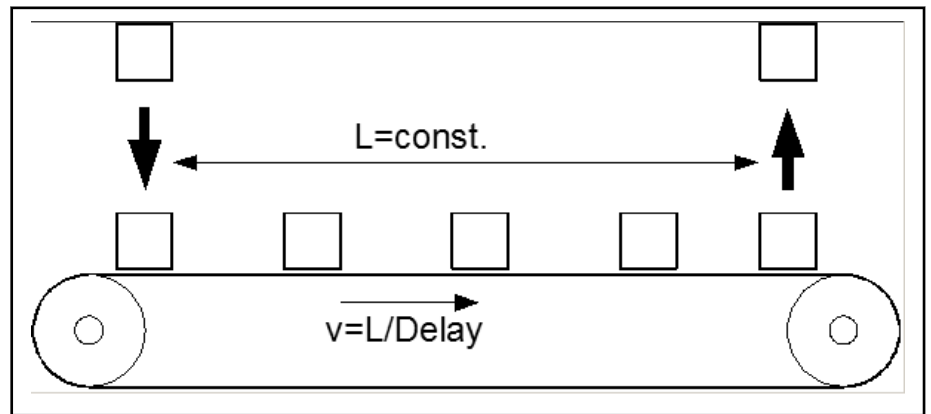


Fig. 7-42:

A conveyor belt runs at a variable velocity to illustrate the dead time alignment. The packages are placed on a fixed point on the belt and removed from the belt once a fixed period has elapsed. The velocity of the belt is set in such a way that every package spends the length of the dead time on the belt. Should the dead time change, then the belt velocity is adjusted accordingly.

Pause

The "Pause" input is only evaluated if the "Enable" input is active. The function block can be stopped via "Pause". In this case, no further values will be stored and "OutputValue" and "ActDelay" remain constant.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated

Configuration

The size of the buffer in "Buffer" affects the accuracy of the output graph. It is possible to create a smaller buffer than 4096 values at "Buffer". Ideally the buffer is larger than the maximum delay required in steps. "BufferSize" can also be less than the delay. In this case, there is not a new value at the "OutputValue" output with each cycle. The resolution, based on the steps, falls.

Required buffer size:

$$\text{"BufferSize"} = \max(\text{"Delay"}) / \text{required resolution}$$

Adequate buffer size:

$$\text{"BufferSize"} = \max(\text{"Delay"})$$

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000001	The "Delay" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000002	The delay given in "Delay" is too large.
INPUT_INVALID_ERROR, 16#0001	16#00000C03	16#00000003	The "Buffer" input is not assigned
INPUT_RANGE_ERROR, 16#0006	16#00000C03	16#00000004	"BufferSize" is too small

Tab. 7-60: Generated error numbers of the IL_TransportDelayType02 function block

7.5.13 IL_ValueLimit

Brief description

This function block limits a value to the range within the limits.

Interface description

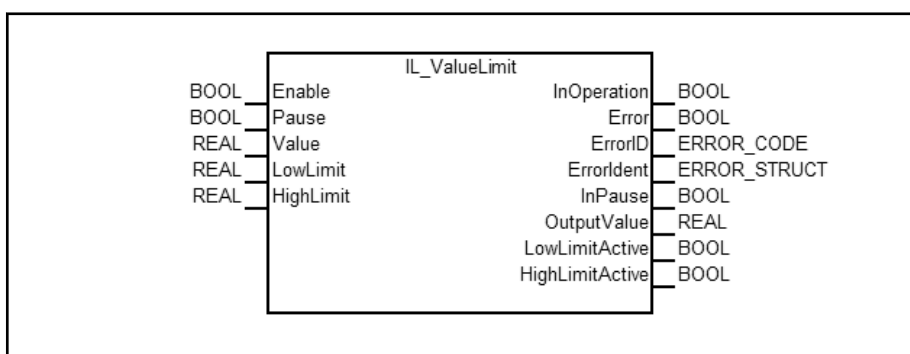


Fig. 7-43: IL_ValueLimit function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, then "Value" is checked for its "LowWindow" and "HighWindow" limits and the "LowWindowActive" and "HighWindowActive" outputs are set accordingly.
	Pause	BOOL	If TRUE, then "InWindow", "LowWindowActive" and "HighWindowActive" remain constant.
	Value	REAL	The current actual value of the process.
	LowLimit	REAL	Lower limit value
	HighLimit	REAL	Upper limit value
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the function block is working and the "InWindow", "LowWindowActive" and "HighWindowActive" outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 385 .
	InPause	BOOL	"Pause" is active and the values at "InWindow", "LowWindowActive" and "HighWindowActive" are frozen.
	OutputValue	REAL	A value limited to the range between "LowLimit" and "HighLimit".

	Name	Type	Comment
	LowLimitActive	BOOL	Active if "Value" ≤ "LowLimit"
	HighLimitActive	BOOL	Active if "Value" ≥ "HighLimit"

Tab. 7-61: IL_ValueLimit function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
LowLimit	REAL	n.def.	<HighLimit	-10.0	Continuous
HighLimit	REAL	>LowLimit	n.def.	10.0	Continuous

Tab. 7-62: Min./max. and default values of the IL_ValueLimit inputs

Functional description

The function block IL_ValueLimit compares the value of the "Value" input with the limit values given in "LowLimit" and "HighLimit" and issues the limited value of "Value" at "OutputValue". The check is carried out continuously when the "Enable" input is set. All inputs are continuously processed.

LowLimitActive, HighLimitActive

The "LowLimitActive" and "HighLimitActive" outputs are set as follows:

- "LowLimitActive" is set if "Value" is less than or equal to "LowLimit".
- "HighLimitActive" is set if "Value" is greater than or equal to "HighLimit".

Pause, InPause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is set, then no comparison is carried out and the "OutputValue", "LowLimitActive" and "HighLimitActive" outputs are not changed.

The "InPause" output is set, when "Pause" is active.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C01	16#00000001	LowLimit is greater than HighLimit.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-63: Generated error numbers of the IL_ValueLimit function block

7.5.14 IL_ValueWindow

Brief description

This function block checks whether a value is within the limits and issues the results of this check. The value is not changed.

Interface description

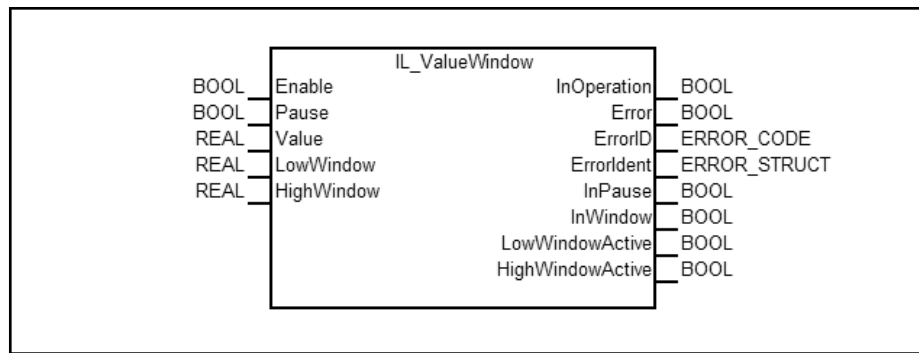


Fig. 7-44: IL_ValueWindow function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, then "Value" is checked for its "LowWindow" and "HighWindow" limits and the "LowWindowActive" and "HighWindowActive" outputs are set accordingly.
	Pause	BOOL	If TRUE, then "InWindow", "LowWindowActive" and "HighWindowActive" remain constant.
	Value	REAL	The current actual value of the process.
	LowWindow	REAL	Lower limit value
	HighWindow	REAL	Upper limit value
VAR_OUTPUT	InOperation	BOOL	If TRUE, then the function block is working and the "InWindow", "LowWindowActive" and "HighWindowActive" outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 387 .
	InPause	BOOL	"Pause" is active and the values at "InWindow", "LowWindowActive" and "HighWindowActive" are frozen.
	InWindow	BOOL	Active if "LowWindow" < "Value" < "HighWindow"
	LowWindowActive	BOOL	Active if "Value" ≤ "LowWindow"
	HighWindowActive	BOOL	Active if "Value" ≥ "HighWindow"

Tab. 7-64: IL_ValueWindow function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
LowWindow	REAL	n.def.	< HighWindow	-10.0	Continuous
HighWindow	REAL	> LowWindow	n.def.	10.0	Continuous

Tab. 7-65: Min./max. and default values of the IL_ValueWindow inputs

Functional description The function block IL_ValueWindow compares the values of the "Value" input with the limit values given in "LowWindow" and "HighWindow". The check is carried out continuously when the "Enable" input is set. All inputs are continuously processed.

InWindow, LowWindowActive, HighWindowActive

The "InWindow", "LowWindowActive" and "HighWindowActive" outputs are set as follows:

- "InWindow" is set if "Value" is greater than "LowWindow" and less than "HighWindow".
- "LowWindowActive" is set if "Value" is less than or equal to "LowWindow".
- "HighWindowActive" is set if "Value" is greater than or equal to "HighWindow".

Pause, InPause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is set, then no comparison is carried out and the "InWindow", "LowWindowActive" and "HighWindowActive" outputs are not changed.

The "InPause" output is set, when "Pause" is active.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C08	16#00000001	LowWindow is greater than HighWindow.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-66: Generated error numbers of the IL_ValueWindow function block

7.6 Filter

7.6.1 Introduction and overview

The RIL_LoopControl provides the following filter function blocks:

Time-discrete	Event-controlled	Description
IL_FIRType01, page 387	IL_FIRType02, page 389	Digital FIR filter for high-pass, low-pass and band-pass usage
IL_IIRType01, page 391	IL_IIRType02, page 394	Digital IIR filter for high-pass, low-pass and band-pass usage
IL_StateVariableType01, page 396	IL_StateVariableType02, page 398	IIR filter (Infinite Impulse Response) second degree for high-pass, low-pass and band-pass use and as band-stop filter

Tab. 7-67: RIL_LoopControl filter

7.6.2 IL_FIRType01

Brief description The IL_FIRType01 function block implements a general time-discrete FIR filter. This digital filter can be used for different high-pass, low-pass and band-pass filters. The function block operates time-discretely.

Interface description

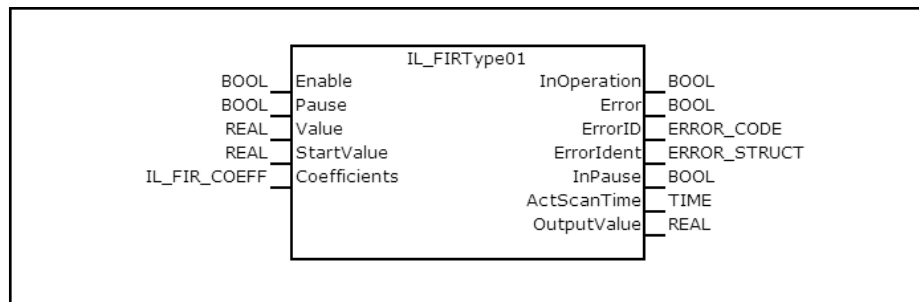


Fig. 7-45: IL_FIRType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	Coefficients	IL_FIR_COEFF	Structure with filter coefficients IL_FIR_COEFF, page 401
VAR_OUTPUT	InOperation	BOOL	If TRUE, the filter is working and the "OutputValue" output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error division, see Error codes, page 389
	InPause	BOOL	"Pause" is active and the value at "OutputValue" is frozen.
	ActScan-Time	TIME	The time elapsed since the latest function block call.
	OutputValue	REAL	The calculated filter value

Tab. 7-68: IL_FIRType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Coefficients	IL_FIR_COEFF				Rising edge at "Enable"

Tab. 7-69: Min./max. and default values of the IL_FIRType01 inputs

Functional description

The IL_FIRType01 function block implements a general time-discrete FIR filter. This digital filter can be used for different high-pass, low-pass and band-pass filters. The function block operates time-discretely.

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. If "Enable" is set and "InPause" is not set, the result is output at the "OutputValue" output.

Coefficients

The filter response is defined via the "Coefficients" input. This data structure consists of the "NumberOfCoeffs" input for the number of coefficients and the "Coeffs" array for the individual coefficients a_0 to a_{15} .

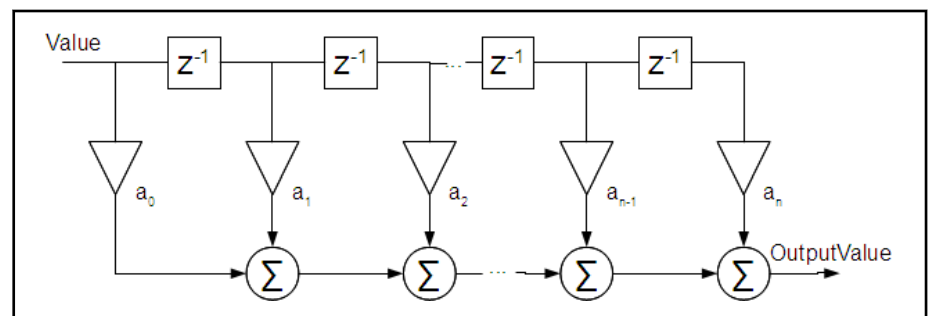


Fig. 7-46: FIR filter scheme

Pause, InPause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal registers are also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Time response

The IL_FIRType01 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "Coefficients" input and the internal registers are loaded with "StartValue". At each call, the function block reads a value from the "Value" input and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0C	16#00000001	The "NumberOfCoeffs" entry of the "Coefficients" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-70: Generated error numbers of the IL_FIRType01 function block

7.6.3 IL_FIRType02

Brief description

The IL_FIRType02 function block implements a general event-controlled FIR filter. This digital filter can be used for different high-pass, low-pass and band-pass filters. This function block operates event-controlled. That means that a new value is processed when the "Step" input is changed.

Interface description

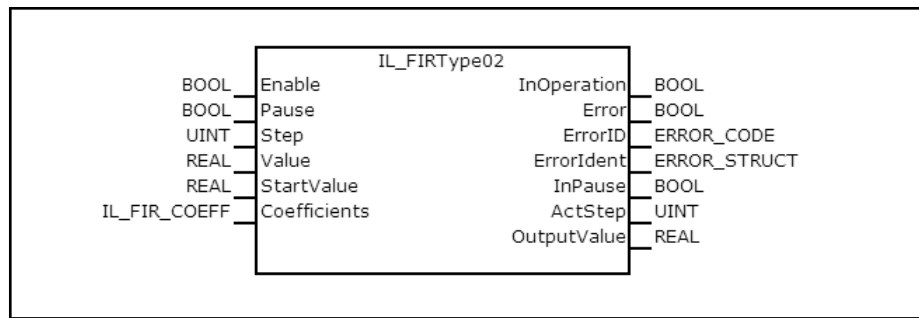


Fig. 7-47: IL_FIRType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Step	UINT	Processing step. A change triggers the processing of a new signal value
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	Coefficients	IL_FIR_COEFF	Structure with filter coefficients IL_FIR_COEFF, page 401
VAR_OUTPUT	InOperation	BOOL	If TRUE, the filter is working and the "OutputValue" output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 391 .
	InPause	BOOL	"Pause" is active and the value at "OutputValue" is frozen.
	ActStep	UINT	Latest step processed
	OutputValue	REAL	The calculated filter value

Tab. 7-71: IL_FIRType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	65535	0	Continuous
Value	REAL	n. def.	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Coefficients	IL_FIR_COEFF				Rising edge at "Enable"

Tab. 7-72: Min./max. and default values of the IL_FIRType02 inputs

Functional description The IL_FIRType02 function block implements a general event-controlled FIR filter. This digital filter can be used for different high-pass, low-pass and band-pass filters. This function block operates event-controlled. That means that a new value is processed when the "Step" input is changed. The amount the value changes at the "Step" input is not important since exactly one filter iteration is executed each time.

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. If "Enable" is set and "InPause" is not set, the result is output at the "OutputValue" output.

Coefficients

The filter response is defined via the "Coefficients" input. This data structure consists of the "NumberOfCoeffs" input for the number of coefficients and the "Coeffs" array for the individual coefficients a_0 to a_{15} .

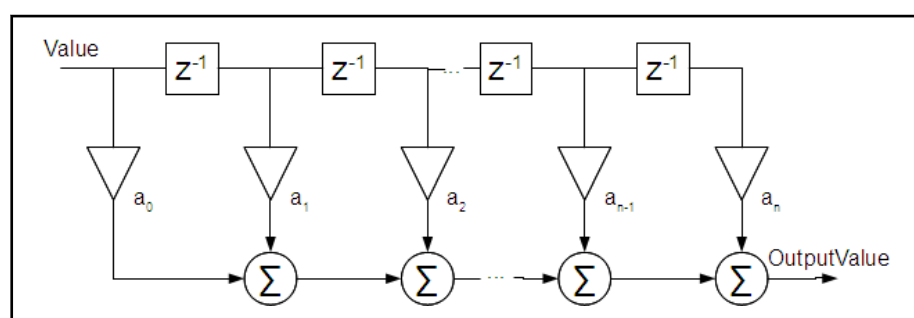


Fig. 7-48: FIR filter scheme

Pause, InPause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal registers are also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Time response The IL_FIRType02 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "Coefficients" input. At each change at the "Step" input, the function block read a new value from the "Value" input and calculates a new value at the "OutputValue" output. If the signal generation is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0C	16#00000001	The "NumberOfCoeffs" entry of the "Coefficients" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-73: Generated error numbers of the IL_FIRType02 function block

7.6.4 IL_IIRType01

Brief description The IL_IIRType01 function block implements a general time-discrete IIR filter (Infinite Impulse Response). This digital filter can be used for example of

Chebyshev, Butterworth and Bessel filters. The function block operates time-discretely.

Interface description

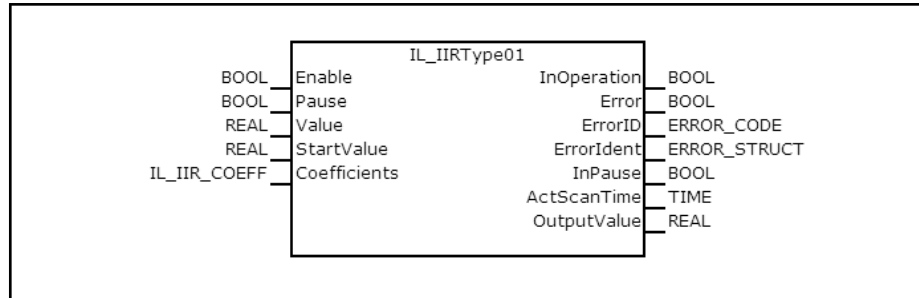


Fig. 7-49: IL_IIRType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	Coefficients	IL_IIR_COEFF	Structure with filter coefficients IL_IIR_COEFF , page 401.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the filter is working and the "OutputValue" output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes , page 393.
	InPause	BOOL	"Pause" is active and the value at "OutputValue" is frozen.
	ActScanTime	TIME	The time elapsed since the latest function block call.
	OutputValue	REAL	The calculated filter value

Tab. 7-74: IL_IIRType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Coefficients	IL_IIR_COEFF				Rising edge at "Enable"

Tab. 7-75: Min./max. and default values of the IL_IIRType01 inputs

Functional description The IL_IIRType01 function block implements a general time-discrete IIR filter. This digital filter can be used for example of Chebyshev, Butterworth and Bessel filters. The function block operates time-discretely.

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. If "Enable" is set and "InPause" is not set, the result is output at the "OutputValue" output.

Coefficients

The filter response is defined via the "Coefficients" input. This data structure consists of the "NumberOfCoeffs" input for the number of coefficients and the "ForwardCoeffs" a_0 to a_{15} array and "BackwardCoeffs" b_1 to b_{15} for the individual coefficients. The "ForwardCoeffs" array contains one entry more than the "BackwardCoeffs" array.

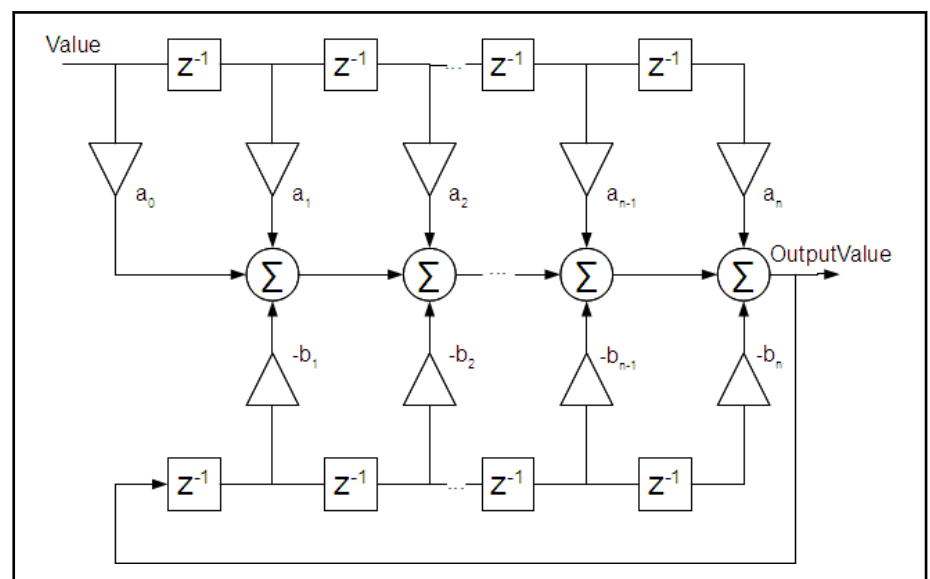


Fig. 7-50: IIR filter scheme

Pause, InPause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal registers are also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Time response The IL_IIRType01 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "Coefficients" input and the internal registers are loaded with "StartValue". At each call, the function block reads a value from the "Value" input and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0C	16#00000001	The "NumberOfCoeffs" entry of the "Coefficients" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-76: Generated error numbers of the IL_IIRType01 function block

7.6.5 IL_IIRType02

Brief description The IL_IIRType02 function block implements a general event-controlled IIR filter. This digital filter can be used for different high-pass, low-pass and band-pass filters. This function block operates event-controlled. That means that a new value is processed when the "Step" input is changed.

Interface description

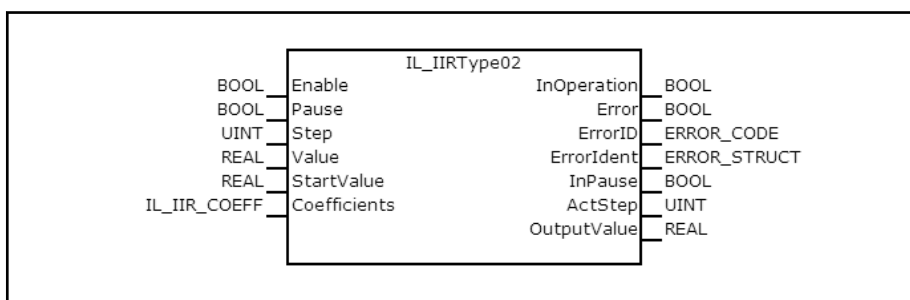


Fig. 7-51: IL_IIRType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Step	UINT	Processing step. A change triggers the processing of a new signal value
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	Coefficients	IL_IIR_COEFF	Structure with filter coefficients IL_IIR_COEFF , page 401.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the filter is working and the "OutputValue" output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes , page 396.
	InPause	BOOL	"Pause" is active and the value at "OutputValue" is frozen.
	ActStep	UINT	Latest step processed
	OutputValue	REAL	The calculated filter value

Tab. 7-77: IL_IIRType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT			0	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
Coefficients	IL_IIR_COEFF				Rising edge at "Enable"

Tab. 7-78: Min./max. and default values of the IL_IIRType02 inputs

Functional description

The IL_IIRType02 function block implements a general event-controlled IIR filter. This digital filter can be used for different high-pass, low-pass and band-pass filters. This function block operates event-controlled. That means that a new value is processed when the "Step" input is changed. The amount the value changes at the "Step" input is not important since exactly one filter iteration is executed each time.

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. If "Enable" is set and "InPause" is not set, the result is output at the "OutputValue" output.

Coefficients

The filter response is defined via the "Coefficients" input. This data structure consists of the "NumberOfCoeffs" input for the number of coefficients and the "ForwardCoeffs" a_0 to a_{15} array and "BackwardCoeffs" b_1 to b_{15} for the individual coefficients. The "ForwardCoeffs" array contains one entry more than the "BackwardCoeffs" array.

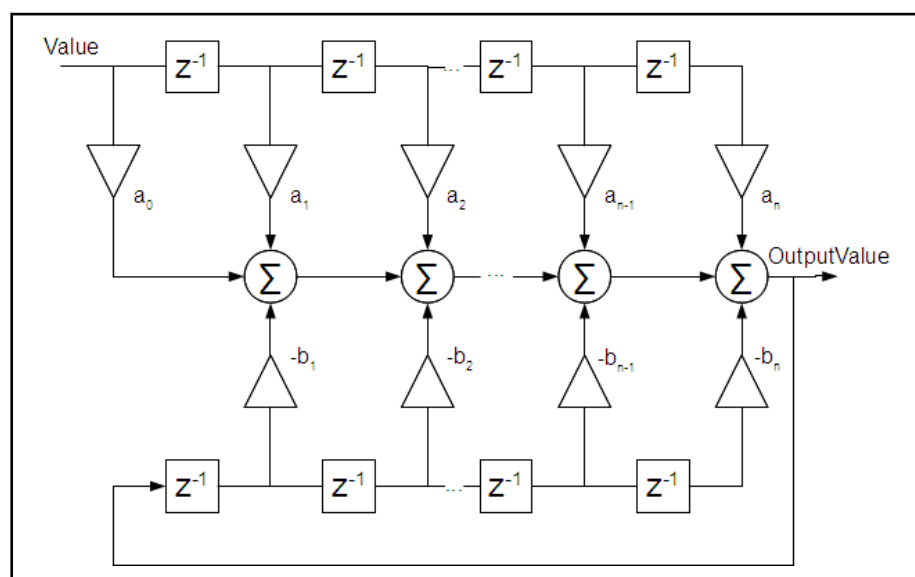


Fig. 7-52: IIR filter scheme

Pause, InPause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal registers are also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Time response The IL_IIRType02 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "Coefficients" input and the internal registers are loaded with "StartValue". At each call, the function block reads a value from the "Value" input and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0C	16#00000001	The "NumberOfCoeffs" entry of the "Coefficients" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-79: Generated error numbers of the IL_IIRType02 function block

7.6.6 IL_StateVariableType01

Brief description The IL_StateVariableType01 function block implements a special time-discrete IIR filter (Infinite Impulse Response) second order. This digital filter can be used as high-pass, band-pass or band-stop filter. The function block operates time-discretely.

Interface description

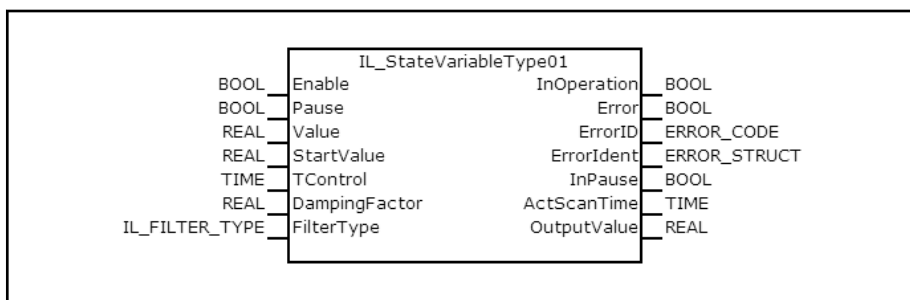


Fig. 7-53: IL_StateVariableType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	TControl	TIME	Time constant
	Damping-Factor	REAL	Damping factor
	FilterType	IL_FILTER_TYPE	Filter type IL_FILTER_TYPE , page 400.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the filter is working and the "OutputValue" output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID

	Name	Type	Comment
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 398 .
	InPause	BOOL	"Pause" is active and the value at "OutputValue" is frozen.
	ActScan-Time	TIME	The time elapsed since the latest function block call.
	OutputValue	REAL	The calculated filter value

Tab. 7-80: IL_StateVariableType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
TControl	TIME	>0s	n.def.	1ms	Continuous
DampingFactor	REAL	0.0		1.0	Continuous
FilterType	IL_FILTER_TYPE			IL_LOWPASS	Rising edge at "Enable"

Tab. 7-81: Min./max. and default values of the IL_StateVariableType01 inputs

Functional description

The IL_StateVariableType01 function block implements a special time-discrete IIR filter (Infinite Impulse Response) second order. This digital filter can be used as high-pass, band-pass or band-stop filter. The function block operates time-discretely.

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the value in "StartValue" is accepted in all internal registers. If "Enable" is set and "InPause" is not set, the result is output at the "OutputValue" output.

TControl, DampingFactor, FilterType

The "FilterType" input indicates the filter to be calculated. The following filter types are available: IL_LOWPASS, IL_HIGHPASS, IL_BANDPASS, IL_BANDREJECT.

Depending on the type of filter, the inputs "TControl" and "DampingFactor" are interpreted:

FilterType	1/TControl	DampingFactor
IL_LOWPASS	Cut-off frequency suppressed above the frequencies	DampingFactor<1: Overshoot, DampingFactor<1: Critical damping
IL_HIGHPASS	Cut-off frequency suppressed below the frequencies	DampingFactor>1: No overshoot Amplitude at cut-off frequency = damping factor 0.5

FilterType	1/TControl	DampingFactor
IL_BANDPASS	Center frequency of the band pass	Amplitude at center frequency = damping factor 0.5
IL_BANDREJECT	Center frequency of the band-stop filter	Cut-off frequency = center frequency*(1±DampingFactor/2)

Tab. 7-82: Interpretation of the input values according to the filter type set

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActScanTime" output continues to be updated.

Time response

The IL_StateVariableType01 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "StartValue" input. At each call, the function block reads a value from the inputs "Value", "TControl" and "DampingFactor" and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000001	The "TControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000008	The "DampingFactor" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000002	The value specified at the "FilterType" input is invalid
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-83: Generated error numbers of the IL_StateVariableType01 function block

7.6.7 IL_StateVariableType02

Brief description

The IL_StateVariableType02 function block implements a special event-controlled IIR filter (Infinite Impulse Response) second order. This digital filter can be used as high-pass, band-pass or band-stop filter. This function block operates event-controlled. That means that a new value is processed when the "Step" input is changed.

Interface description

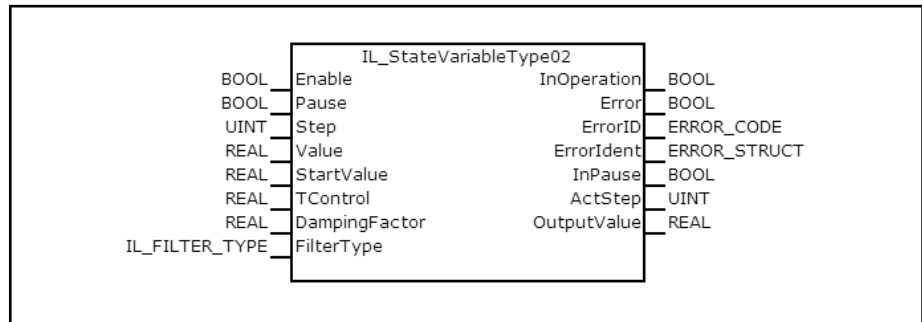


Fig. 7-54: IL_StateVariableType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	If TRUE, "OutputValue" is set to the value calculated with "StartValue".
	Pause	BOOL	Enables the Pause mode if TRUE
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
	TControl	REAL	Time constant
	Damping-Factor	REAL	Damping factor
	FilterType	IL_FILTER_TYPE	Filter type IL_FILTER_TYPE , page 400.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the filter is working and the "OutputValue" output is valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes , page 400.
	InPause	BOOL	"Pause" is active and the value at "OutputValue" is frozen.
	ActStep	UINT	Step to which "OutputValue" refers.
	OutputValue	REAL	The calculated filter value

Tab. 7-84: IL_StateVariableType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
StartValue	REAL	n.def.	n.def.	0.0	Rising edge at "Enable"
TControl	REAL	>0.0	n.def.	1	Continuous
DampingFactor	REAL	0.0		1.0	Continuous
FilterType	IL_FILTER_TYPE			IL_LOWPASS	Rising edge at "Enable"

Tab. 7-85: Min./max. and default values of the IL_StateVariableType02 inputs

Functional description

The IL_StateVariableType02 function block implements a special event-controlled IIR filter (Infinite Impulse Response) second order. This digital filter can be used as high-pass, band-pass or band-stop filter. This function block operates event-controlled. That means that a new value is processed when the "Step" input is changed.

Value, StartValue, OutputValue

The "Value" input contains the input value of the filter. This value is processed at each call if "InOperation" is set. At a rising edge at "Enable", the

value in "StartValue" is accepted in all internal registers. If "Enable" is set and "InPause" is not set, the result is output at the "OutputValue" output.

TControl, DampingFactor, FilterType

The "FilterType" input indicates the filter to be calculated. The following filter types are available: IL_LOWPASS, IL_HIGHPASS, IL_BANDPASS, IL_BANDREJECT.

Depending on the type of filter, the inputs "TControl" and "DampingFactor" are interpreted:

FilterType	1/TControl	DampingFactor
IL_LOWPASS	Cut-off frequency suppressed above the frequencies	DampingFactor<1: Overshoot DampingFactor<1: Critical damping DampingFactor>1: No overshoot
IL_HIGHPASS	Cut-off frequency suppressed below the frequencies	
IL_BANDPASS	Center frequency of the band pass	Band width = 2*DampingFactor/TControl
IL_BANDREJECT	Center frequency of the band-stop filter	

Tab. 7-86: Interpretation of the input values according to the filter type set

Pause

The "Pause" input is only evaluated when the "Enable" input is active. If "Pause" is TRUE, it is not calculated and the value of "OutputValue" is not changed. The internal integrator is also frozen.

The "InPause" output is set, when "Pause" is active.

The "ActStep" output continues to be updated.

Time response

The IL_StateVariableType02 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "StartValue" input. At each change at the "Step" input, the function block read a new value from the inputs "Value", "TControl" and "DampingFactor" and calculates a new value at the "OutputValue" output. If the filter is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000001	The "TControl" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000008	The "DampingFactor" input is invalid.
INPUT_RANGE_ERROR, 16#0006	16#00000C02	16#00000002	The value specified at the "FilterType" input is invalid
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-87: Generated error numbers of the IL_StateVariableType02 function block

7.6.8 IL_FILTER_TYPE

The enumeration type IL_FILTER_TYPE defines several basic filter types.

Element	Value	Description
IL_LOWPASS	1	Low-pass
IL_HIGHPASS	2	High-pass

Element	Value	Description
IL_BANDPASS	3	Band-pass
IL_BANDREJECT	4	Band-stop filter

Tab. 7-88: Elements of the IL_FILTER_TYPE enumeration type

Usage in

- [IL_StateVariableType01](#), page 396,
- [IL_StateVariableType02](#), page 398

7.6.9 IL_FIR_COEFF

The IF_FIR_COEFF data type is used to set the filter coefficients.

Structure element	Type	Min. value	Max. value	Default	Description
Coeffs	ARRAY[0..15] OF LREAL			16(0.0)	Filter coefficients
NumberOfOeffs	UINT	1	15	1	Number of filter coefficients

Tab. 7-89: IL_FIR_COEFF data structure

Usage in

- [IL_FIRType01](#), page 387,
- [IL_FIRType02](#), page 389.

7.6.10 IL_IIR_COEFF

The IF_IIR_COEFF data type is used to set the filter coefficients.

Structure element	Type	Min. value	Max. value	Default	Description
ForwardCoeffs	ARRAY[0..15] OF LREAL			16(0.0)	Filter coefficients of the forward path
BackwardCoeffs	ARRAY[1..15] OF LREAL			15(0.0)	Filter coefficients of the feedback loop
NumberOfOeffs	UINT	1	15	1	Number of filter coefficients

Tab. 7-90: IL_IIR_COEFF data structure

Usage in

- [IL_IIRType01](#), page 391,
- [IL_IIRType02](#), page 394.

7.7 Generators (signal generators)

7.7.1 Introduction and overview

This section contains the RIL_LoopControl signal generators. Generators are signal sources.

The RIL_LoopControl provides the following signal generators:

Time-discrete	Event-controlled	Description
Generators		
IL_NoiseGeneratorType01, page 402		Pseudorandom number and noise generator
IL_NoiseGeneratorType02, page 404		Noise generator for white noise
IL_PRBSGeneratorType01, page 405	IL_PRBSGeneratorType02, page 407	Generator for pseudorandom binary numbers
Generators/SignalGenerators		
IL_AngleGeneratorType01, page 410	IL_AngleGeneratorType02, page 411	Angle generator
IL_PhaseController, page 412		Phase controller
IL_SignalGeneratorType01, page 414	IL_SignalGeneratorType02, page 418	Signal generator
IL_AmplitudeControllerType01, page 422	IL_AmplitudeControllerType02, page 425	Amplitude controller

Tab. 7-91: RIL_LoopControl signal generators

The signal generator function blocks IL_NoiseGeneratorType01, IL_NoiseGeneratorType02 and IL_PRBSGeneratorType0x are stand-alone function blocks and can be directly used. They have a "OutputValue" output, but no "Value" input.

The function blocks IL_AngleGeneratorType0x, IL_PhaseController, IL_SignalGeneratorType0x and IL_AmplitudeControllerType0x belong together and can be interconnected as follows:

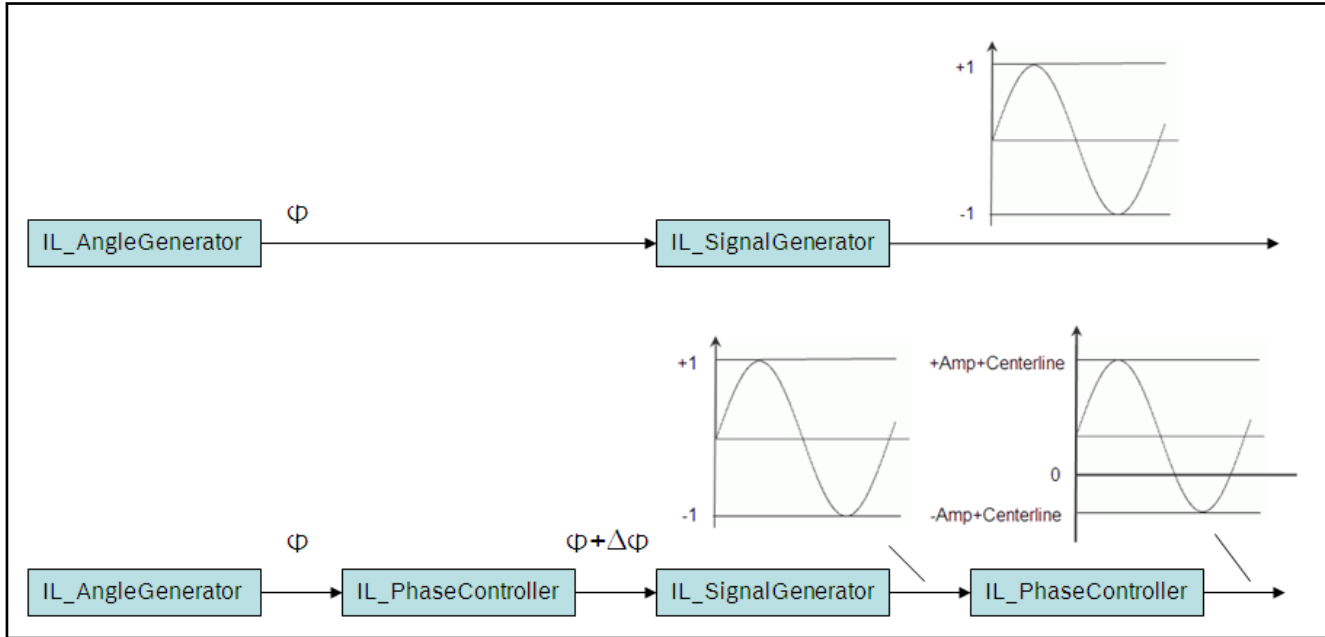


Fig. 7-55: Connection examples on signal generator function blocks

7.7.2 IL_NoiseGeneratorType01

Brief description This function block generates an equally-distributed noise signal with selectable amplitude based on a pseudorandom number.

Interface description

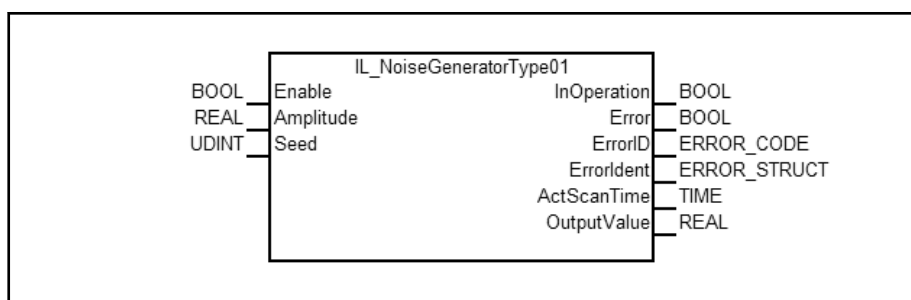


Fig. 7-56: IL_NoiseGeneratorType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Amplitude	REAL	Amplitude of the noise signal
	Seed	UDINT	Initial value, 0 for CPU tick
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 403 .
	ActScanTime	TIME	Measured call cycle time.
	OutputValue	REAL	Generated pseudorandom number.

Tab. 7-92: IL_NoiseGeneratorType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Amplitude	REAL	0.0	n.def.	1.0	Continuous
Seed	UDINT	0	2 ³² -1	0	Rising edge at Enable

Tab. 7-93: Min./max. and default values of the IL_NoiseGeneratorType01 inputs

Functional description

The IL_NoiseGeneratorType01 function block generates an equally-distributed noise signal based on pseudorandom numbers. At a rising edge at "Enable", the value is read at the "Seed" input. If "Seed" is 0, the current CPU counter is used for initialization. All other values at "Seed" are directly used as initial value. If "InOperation" is set, the block generates a new value between 0.0 and "Amplitude" at "OutputValue" with each call. This function block should be called cyclically. The measured call cycle is output in "ActScanTime". "OutputValue" contains the noise signal generated.

The random numbers created are equally-distributed within the range between 0 and "Amplitude".

It is processed continuously if the "Enable" input is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C06	16#00000001	"Amplitude" is smaller than 0.0
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-94: Generated error numbers of the IL_NoiseGeneratorType01 function block

7.7.3 IL_NoiseGeneratorType02

Brief description The IL_NoiseGeneratorType02 function block generates a noise signal around a specified mean value that is Gaussian-distributed in the amplitude. This signal is a white noise which means that all frequencies are in the same amplitude. Additionally, it is DC-free. This function block operates time-discretely.

Interface description

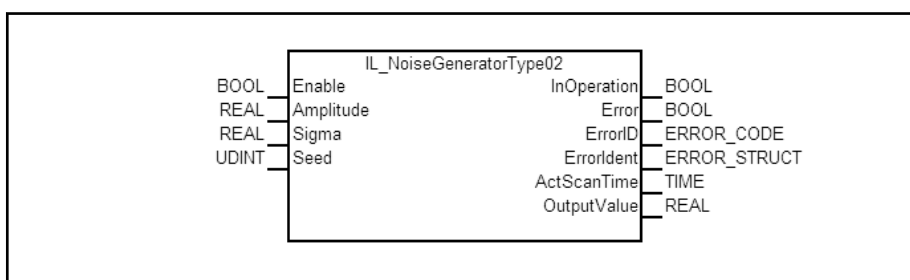


Fig. 7-57: IL_NoiseGeneratorType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Amplitude	REAL	Mean signal amplitude
	Sigma	REAL	Default signal deviation
	Seed	UDINT	Initial value, 0 for CPU tick
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 405 .
	ActScan-Time	TIME	Measured call cycle time.
	OutputValue	REAL	Generated noise signal.

Tab. 7-95: IL_NoiseGeneratorType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Amplitude	REAL	0.0	n.def.	1.0	Rising edge at Enable

Name	Type	Min. value	Max. value	Default value	Effective
Sigma	REAL	>0.0		1.0	Rising edge at Enable
Seed	UDINT	0	$2^{32}-1$	0	Rising edge at Enable

Tab. 7-96: Min./max. and default values of the IL_NoiseGeneratorType02 inputs

Functional description

The IL_NoiseGeneratorType02 function block generates a noise signal that is Gaussian-distributed in the amplitude. This signal is a white noise which means that all frequencies are in the same amplitude. Additionally, it is DC-free. This function block operates time-discretely.

Amplitude, Sigma:

The "Amplitude" input defines the mean amplitude of the generated signal also called μ or expectation window. The generated signal fluctuates around this value. The "Sigma" input indicates the default variation, also called σ .

This results in the following value distribution:

68.3 % in the interval $\mu \pm \sigma$

95.4 % in the interval $\mu \pm 2\sigma$

99.7 % in the interval $\mu \pm 3\sigma$

Seed:

This input is used as initial value for the internal random generator. It can be generally omitted.

Time response:

The IL_NoiseGeneratorType02 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the inputs "Amplitude", "Sigma" and "Seed". The function block generates a new value at the "OutputValue" output at each call. If the signal generation is active, the "In-Operation" output is set. If an error occurs, the "Error" output is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C0B	16#00000001	"Sigma" input is smaller or equal to 0.0.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#0000FFFF	Runtime error in function block: Invalid state of the state machine.

Tab. 7-97: Generated error numbers of the IL_NoiseGeneratorType02 function block

7.7.4 IL_PRBSGeneratorType01

Brief description

The IL_PRBSGeneratorType01 function block generates the pseudo noise signal. This signal is almost a white noise which means that all frequencies are in the same amplitude. Additionally, it is almost DC-free. This function block operates time-discretely.

Interface description

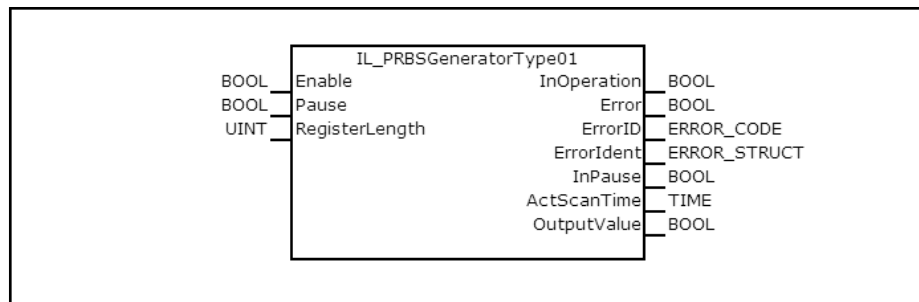


Fig. 7-58: IL_PRBSGeneratorType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Aborts the signal generation
	Register- Length	UINT	Length of the internal shift register
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 407 .
	InPause	BOOL	Processing interrupted
	ActScan- Time	TIME	Measured call cycle time.
	OutputValue	BOOL	Generated pseudorandom number.

Tab. 7-98: IL_PRBSGeneratorType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
RegisterLength	UINT	4	32	4	Rising edge at Enable

Tab. 7-99: Min./max. and default values of the IL_PRBSGeneratorType01 inputs

Functional description

The IL_PRBSGeneratorType01 function block generates the pseudo noise signal. This signal is almost a white noise which means that all frequencies are in the same amplitude. Additionally, it is almost DC-free. This function block operates time-discretely.

The generated signal is repeated after n cycles. The number of cycles depends on the register depth.

RegisterLength:

"RegisterLength" sets the depth of the internal shift register used for signal generation. The register depth does not influence the signal quality generated. A higher register depth generates a longer cycle before the signal is repeated.

The following table shows the signal period depending on the "RegisterLength":

RegisterLength	Period	RegisterLength	Period
		17	131071
2	3	18	262143
3	7	19	524287
4	15	20	1048575
5	31	21	2097151
6	63	22	4194303
7	127	23	8388607
8	255	24	16777215
9	511	25	33554431
10	1023	26	67108863
11	2047	27	134217727
12	4095	28	268435455
13	8191	29	536870911
14	16383	30	1073741823
15	32767	31	2147483647
16	65535	32	4294967295

"RegisterLength" may only contain the values listed in the table.

Pause, InPause:

The function block processing can be temporarily interrupted with the "Pause" input. If "Pause" is active, the "InPause" output is set. No new signal values are generated at the "OutputValue" output in this mode.

Time response:

The IL_PRBSGeneratorType01 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "RegisterLength" input. The function block generates a new value at the "OutputValue" output at each call. If the signal generation is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C0A	16#00000001	The "RegisterLength" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#0000FFFF	Runtime error in function block: Invalid state of the state machine.

Tab. 7-100: Generated error numbers of the IL_PRBSGeneratorType01 function block

7.7.5 IL_PRBSGeneratorType02

Brief description The IL_PRBSGeneratorType02 function block generates a pseudo noise signal. This signal is almost a white noise which means that all frequencies are

in the same amplitude. Additionally, it is almost DC-free. This function block operates event-controlled. That means that a new value is generated if the "Step" input is changed.

Interface description

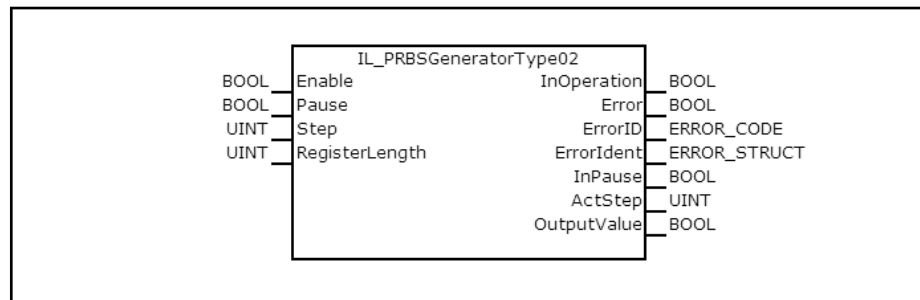


Fig. 7-59: IL_PRBSGeneratorType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Aborts the signal generation
	Step	UINT	Processing step. A change triggers a new signal value
	Register-Length	UINT	Length of the internal shift register
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 409 .
	InPause	BOOL	Processing interrupted
	ActStep	UINT	Latest step processed
	OutputValue	BOOL	Generated pseudo noise signal

Tab. 7-101: IL_PRBSGeneratorType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous if "Enable" is set
Step	UINT	0	65535	0	Continuous if "Enable" is set
RegisterLength	UINT	4	32	4	Rising edge at Enable

Tab. 7-102: Min./max. and default values of the IL_PRBSGeneratorType02 inputs

Functional description

The IL_PRBSGeneratorType02 function block generates a pseudo noise signal. This signal is almost a white noise which means that all frequencies are in the same amplitude. Additionally, it is almost DC-free. This function block

operates event-controlled. That means that a new value is generated if the "Step" input is changed.

The generated signal is repeated after n cycles. The number of cycles depends on the register depth.

RegisterLength:

"RegisterLength" sets the depth of the internal shift register used for signal generation. The register depth does not influence the signal quality generated. A higher register depth generates a longer cycle before the signal is repeated.

The following table shows the signal period depending on the "RegisterLength":

RegisterLength	Period	RegisterLength	Period
		17	131071
2	3	18	262143
3	7	19	524287
4	15	20	1048575
5	31	21	2097151
6	63	22	4194303
7	127	23	8388607
8	255	24	16777215
9	511	25	33554431
10	1023	26	67108863
11	2047	27	134217727
12	4095	28	268435455
13	8191	29	536870911
14	16383	30	1073741823
15	32767	31	2147483647
16	65535	32	4294967295

"RegisterLength" may only contain the values listed in the table.

Pause, InPause:

The function block processing can be temporarily interrupted with the "Pause" input. If "Pause" is active, the "InPause" output is set. No new signal values are generated at the "OutputValue" output in this mode.

Time response:

The IL_PRBSGeneratorType02 function block operates asynchronously. After the start due to a rising edge at the "Enable" input, it reads the "RegisterLength" input. The function block generates a new value at the "OutputValue" output each time the "Step" input is changed. If the signal generation is active, the "InOperation" output is set. If an error occurs, the "Error" output is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C0A	16#00000001	The "RegisterLength" input is invalid.
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#0000FFFF	Runtime error in function block: Invalid state of the state machine.

Tab. 7-103: Generated error numbers of the IL_PRBSGeneratorType02 function block

7.7.6 IL_AngleGeneratorType01

Brief description The function blocks generate an angle between 0.0 and 360.0 degree using the specified frequency.

Interface description

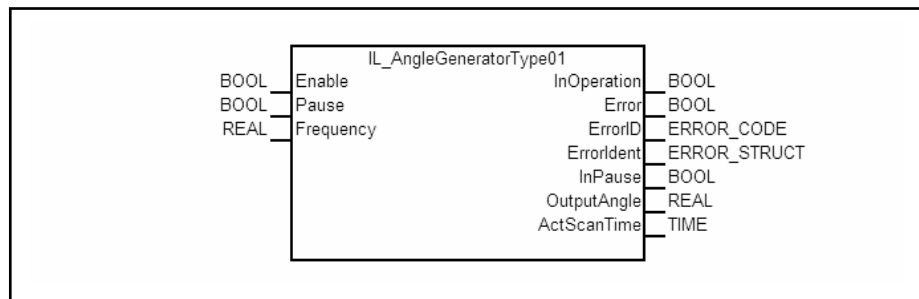


Fig. 7-60: IL_AngleGeneratorType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Pauses processing
	Frequency	REAL	Frequency in Hz
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 411 .
	InPause	BOOL	Pauses active processing
	OutputAngle	REAL	Generated angle
	ActScan-Time	TIME	Measured call cycle time.

Tab. 7-104: IL_AngleGeneratorType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Frequency	REAL	0.0	1/(2*cycle time)	1.0	Continuous

Tab. 7-105: Min./max. and default values of the IL_AngleGeneratorType01 inputs

Functional description The IL_AngleGeneratorType01 function block generates an angle between 0 and 360 degrees with the frequency specified in Hz at the "Frequency" input.

If "InOperation" is set, the block generates a new value between 0.0 and 360.0 at "OutputAngle" with each call. This function block should be called cyclically. The measured call cycle is output in "ActScanTime".

The created angular values are between 0 and 360 degrees.

It is processed continuously if the "Enable" input is set. The "Pause" input temporarily pauses the processing.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0D	16#00000001	"Frequency" is lower than 0.0 or higher than 1/ (2*ActScanTime)
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-106: Generated error numbers of the IL_AngleGeneratorType01 function block

7.7.7 IL_AngleGeneratorType02

Brief description The function blocks generate an angle between 0.0 and 360.0 degree using the specified frequency.

Interface description

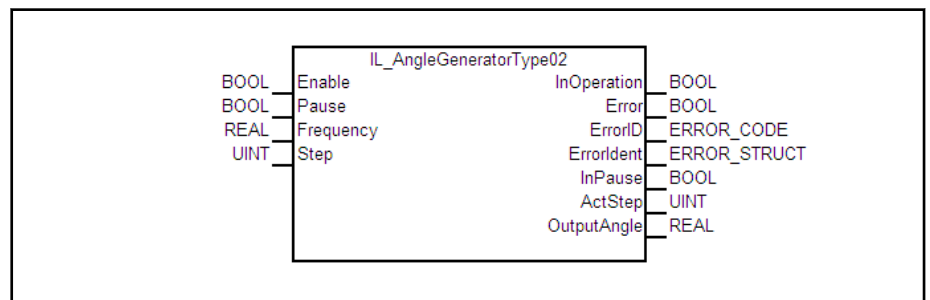


Fig. 7-61: IL_AngleGeneratorType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Pauses processing
	Frequency	REAL	Frequency in 1/steps
	Step	UINT	Processing step
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 412 .
	InPause	BOOL	Pauses active processing
	ActStep	UINT	Measured call cycle time.
	OutputAngle	REAL	Generated angle

Tab. 7-107: IL_AngleGeneratorType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Frequency	REAL	0.0	0.5	0.1	Change at "Step"
Step	BOOL	0	65535	0	Continuous

Tab. 7-108: Min./max. and default values of the IL_AngleGeneratorType02 inputs

Functional description

The IL_AngleGeneratorType01 function block generates an angle between 0 and 360 degrees with the frequency specified in 1/Step at the "Frequency" input. If "InOperation" is set, the block generates a new value between 0.0 and 360.0 at "OutputAngle" with each call. This function block should be called cyclically. The frequency specified at the "Frequency" input does not refer to the steps at the "Step" input. Thus, it has to be lower than 0.5. Calculation only takes place if the "Step" input changes.

The created angular values are between 0 and 360 degrees.

The processing is controlled via the "Step" input when the "Enable" input is set. The "Pause" input temporarily pauses the processing.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0D	16#00000001	"Frequency" is lower than 0.0 or higher than 0.5
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-109: Generated error numbers of the IL_AngleGeneratorType02 function block

7.7.8 IL_PhaseController

Brief description

This function block corrects the phase of an angle via a control loop.

Interface description

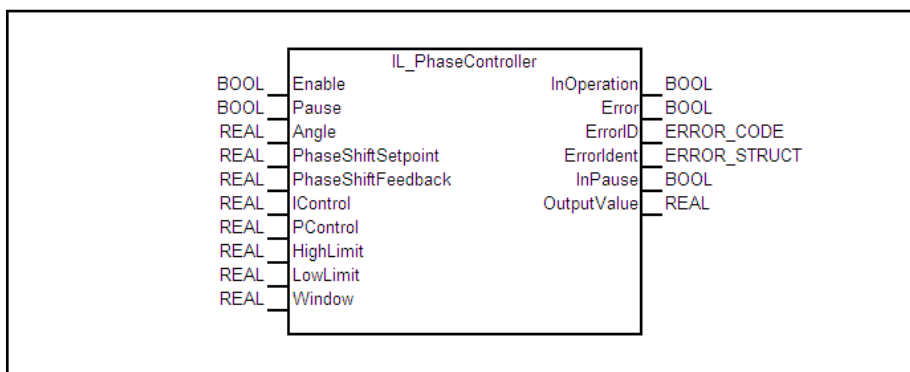


Fig. 7-62: IL_PhaseController function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Pauses processing
	Angle	REAL	Input angle
	PhaseShift-Setpoint	REAL	Reference variable for phase control
	PhaseShift-Feedback	REAL	Feedback value for control loop
	IControl	REAL	I-value of the controller
	PControl	REAL	P-value of the controller
	HighLimit	REAL	Upper correction limit in degrees
	LowLimit	REAL	Lower correction limit in degrees
	Window	REAL	Control window
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 414 .
	InPause	BOOL	Pauses active processing
	OutputValue	REAL	Controlled angle

Tab. 7-110: IL_PhaseController function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Angle	REAL	0.0	360.0	0.0	Continuous
PhaseShiftSetpoint	REAL			0.0	Continuous
PhaseShiftFeedback	REAL			0.0	Continuous
IControl	REAL	0.0		0.0	Continuous
PControl	REAL	0.0		0.0	Continuous
HighLimit	REAL	0.0	180.0	10.0	Continuous
LowLimit	REAL	-180.0	0.0	-10.0	Continuous
Window	REAL	0.0		5.0	Continuous

Tab. 7-111: Min./max. and default values of the IL_PhaseController inputs

Hidden outputs

To reduce the complexity of the function block, not all internal values are available at the outputs. These values can be directly read at the function block. Write-access is not possible.

Variable	Data type	Description
ActAngle	REAL	Calculated target zero
IpartCorrection	REAL	Correction value of the PI-controller
PhaseShift	REAL	Calculated phase offset

Tab. 7-112: Internal values of the function block read via properties

Functional description

The IL_PhaseController function block shifts the input angle "Angle" by a correction value and outputs it at "OutputAngle". The correction value is specified by a PI-controller controlling once per cycle in the zero crossing when the difference between "PhaseShiftSetPoint" and "PhaseShiftFeedback" is higher than "Window". The correction value is limited to a value between "LowLimit" and "HighLimit".

The control settings "PControl" and "IControl" refer to one cycle. One cycle starts if the "Angle" input jumps from 360 to 0 degrees.

The created angular values are between 0 and 360 degrees.

It is processed continuously if the "Enable" input is set.

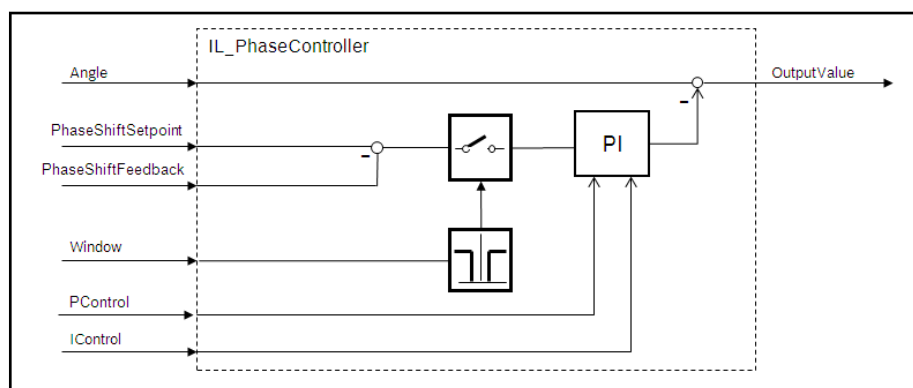


Fig. 7-63: Signal flow of the IL_PhaseController function block

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000001	"Angle" is smaller than 0° or higher than 360°
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000002	"PhaseShiftSetpoint" is smaller than -180 or higher than 180
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000003	"PhaseShiftFeedback" is smaller than -180 or higher than 180
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000004	"IControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000005	"PControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000006	"HighLimit" is smaller than "LowLimit"
INPUT_RANGE_ERROR, 16#0001	16#00000C0E	16#00000007	"Window" is smaller than 0
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-113: Generated error numbers of the IL_PhaseController function block

7.7.9 IL_SignalGeneratorType01

Brief description This function block generates different signal shaped using an input angle.

Interface description

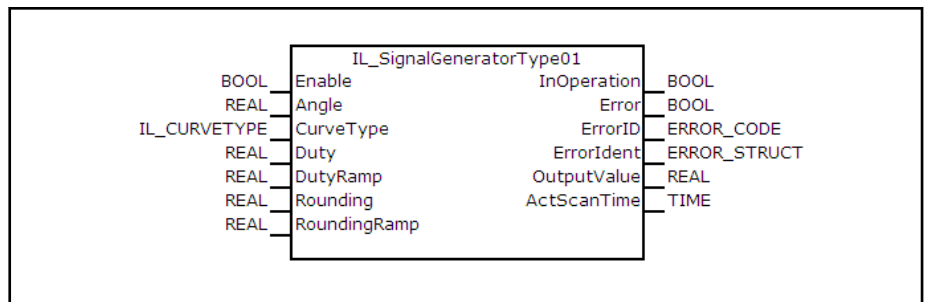


Fig. 7-64: IL_SignalGeneratorType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Angle	REAL	angle
	CurveType	IL_CURVETYPE	Generated signal shape IL_CURVETYPE, page 429
	Duty	REAL	Sample ratio in % (50=symmetric) Permitted value range: - Sinusoidal: 25-75 - Double_Sine: 10-90 - Triangular: 10-90
	DutyRamp	REAL	Changing the sample ratio in %/ms
	Rounding	REAL	Rounding the signal in % (only active for signal shape "triangle")
	Rounding-Ramp	REAL	Changing the rounding in %/ms
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 418 .
	OutputValue	REAL	Generated signal shape
	ActScan-Time	TIME	Measured cycle time

Tab. 7-114: IL_SignalGeneratorType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Angle	REAL	0.0	360.0	0.0	Continuous
CurveType	IL_CURVETYPE			SINUSOIDAL	Continuous
Duty	REAL	10.0	90.0	50.0	Continuous
DutyRamp	REAL	0.0	100.0	0.0001	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
Rounding	REAL	0.0	100.0	0.0	Continuous
RoundingRamp	REAL	0.0	100.0	0.0001	Continuous

Tab. 7-115: Min./max. and default values of the IL_SignalGeneratorType01 inputs

Functional description

The IL_SignalGeneratorType01 function block generates the signal shape set at the "CurveType" input based on the angle applied at the "Angle" input. The sample ratio can be set via the "Duty" input. If "InOperation" is set, the function block generates a new value between -1.0 and 1.0 at the "OutputValue" output.

It is processed continuously if the "Enable" input is set.

CurveType

The "CurveType" input determines the generated signal shape. The following signal shapes are implemented:

- SINUSOIDAL: Function similar to sinusoidal with longest consistently increasing and decreasing sections possible

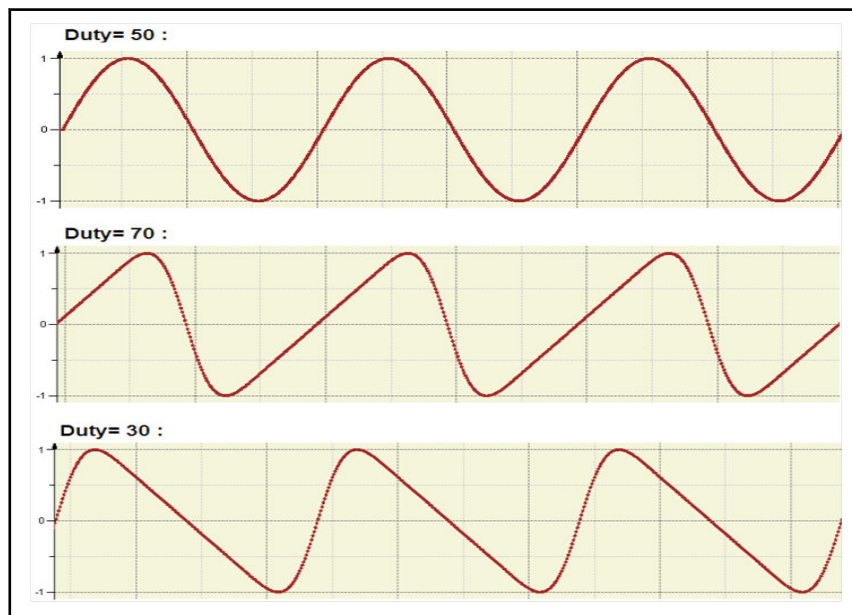


Fig. 7-65: Signal shape: sinusoidal

- DOUBLE_SINE: Double sine

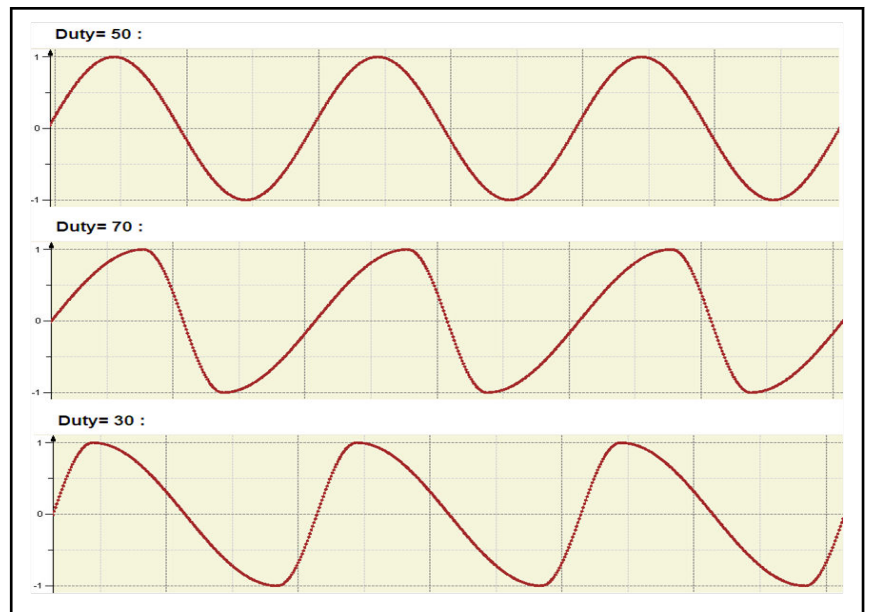


Fig. 7-66: Signal shape: double sine

- TRIANGULAR: Triangle signal

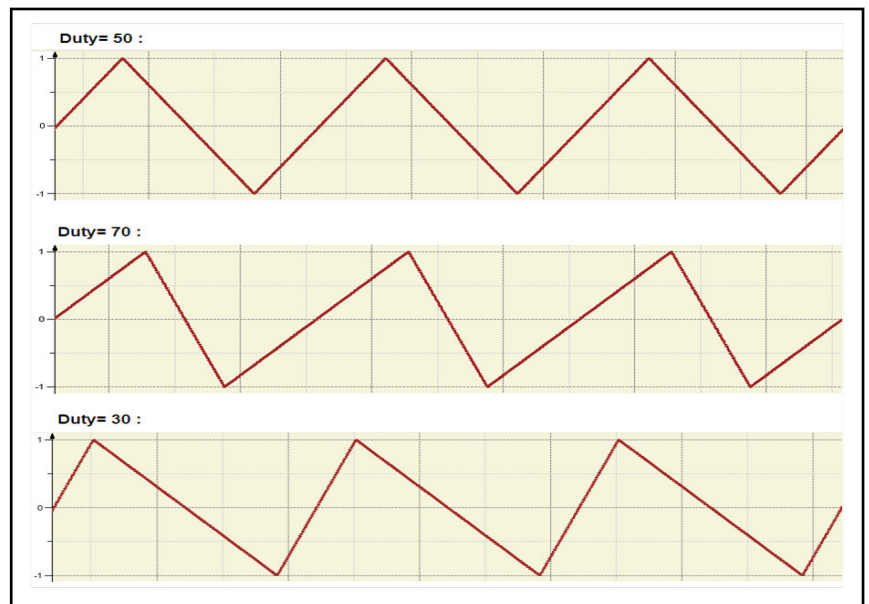


Fig. 7-67: Signal shape: Triangle

Duty, DutyRamp

The "Duty" input determined the sample ratio of the signal. Actually, the 180° mark is moved. With a sample ratio of 50%, the signal is symmetrical. Depending on the signal shape, "Duty" is limited.

The "DutyRamp" input specifies the change rate of the sampling ratio in %/ms if the value is changed at the "Duty" input.

Rounding, RoundingRamp

The "Rounding" input is only enabled if the signal shape "TRIANGULAR" is active. It indicates the rounding of the triangle in %. 0 means no rounding.

The "RoundingRamp" input specifies the change rate of the rounding in %/ms if the "Rounding" input is changed.

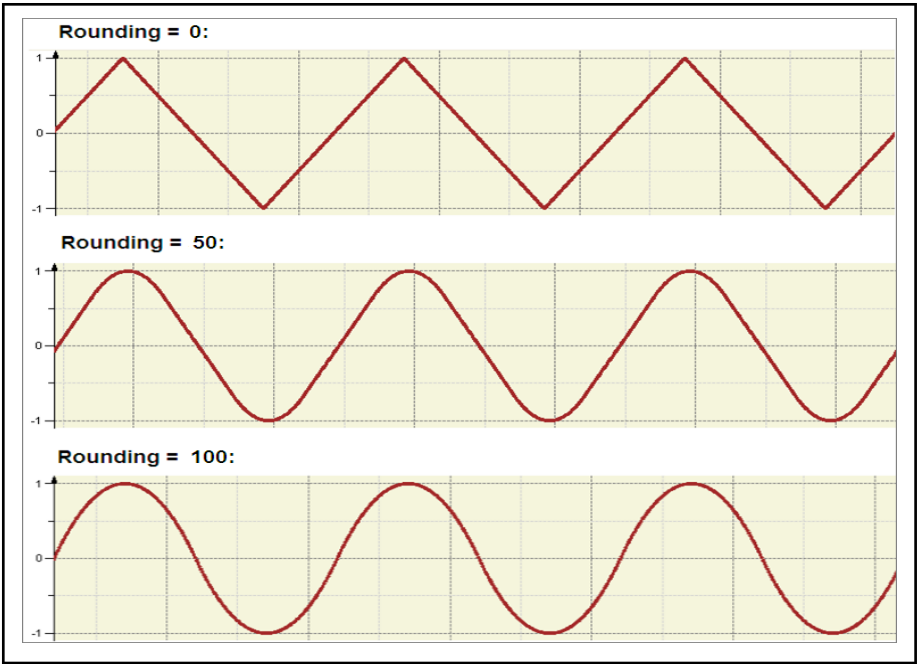


Fig. 7-68: Effects of the "Rounding" input on the signal shape

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000001	"Angle" is lower than 0.0 or higher than 360.0
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000002	"CurveType" is invalid
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000003	"Duty" is outside the permitted range
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000004	"DutyRamp" is smaller than 0.0001
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000005	"Rounding" is negative or higher than 100
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000006	"RoundingRamp" is smaller than 0.0001
STATE_MACHINE_ERROR,16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-116: Generated error numbers of the IL_SignalGeneratorType01 function block

7.7.10 IL_SignalGeneratorType02

Brief description This function block generates different signal shaped using an input angle.
Interface description

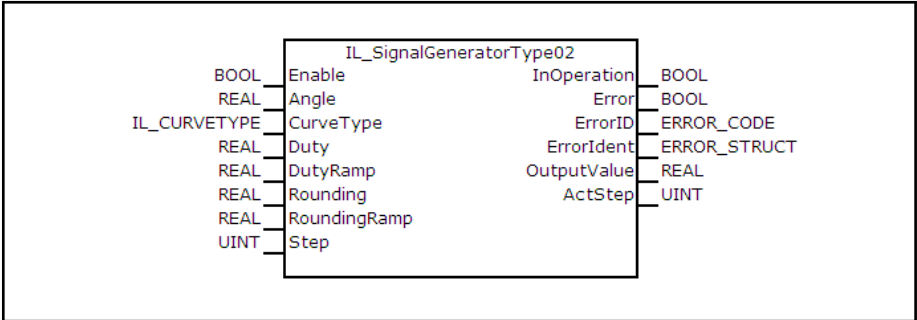


Fig. 7-69: IL_SignalGeneratorType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Angle	REAL	angle
	CurveType	IL_CURVETYPE	Generated signal shape IL_CURVETYPE, page 429
	Duty	REAL	Sample ratio in % (50=symmetric) - Sinusoidal: 25-75 - Double_Sine: 10-90 - Triangular: 10-90
	DutyRamp	REAL	Changing the sample ratio in %/Step
	Rounding	REAL	Rounding the signal in % (only active for signal shape "triangle")
	Rounding-Ramp	REAL	Changing the rounding in %/Step
	Step	UINT	Processing step
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 422 .
	OutputValue	REAL	Generated signal shape
	ActStep	UINT	Latest processing step executed

Tab. 7-117: IL_SignalGeneratorType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Angle	REAL	0.0	360.0	0.0	Change at "Step"
CurveType	IL_CURVETYPE			SINUSOIDAL	Change at "Step"
Duty	REAL	10.0	90.0	50.0	Change at "Step"
DutyRamp	REAL	0.0	100.0	0.0001	Change at "Step"
Rounding	REAL	0.0	100.0	0.0	Change at "Step"
RoundingRamp	REAL	0.0	100.0	0.0001	Change at "Step"

Tab. 7-118: Min./max. and default values of the IL_SignalGeneratorType02 inputs

Functional description

The IL_SignalGeneratorType02 function block generates the signal shape set at the "CurveType" input based on the angle applied at the "Angle" input. The sample ratio can be set via the "Duty" input. If "InOperation" is set, the function block generates a new value between -1.0 and 1.0 at the "OutputValue" output whenever "Step" is changed.

It is processed continuously if the "Enable" input is set.

CurveType

The "CurveType" input determines the generated signal shape. The following signal shapes are implemented:

- SINUSOIDAL: Function similar to sinusoidal with longest consistently increasing and decreasing sections possible

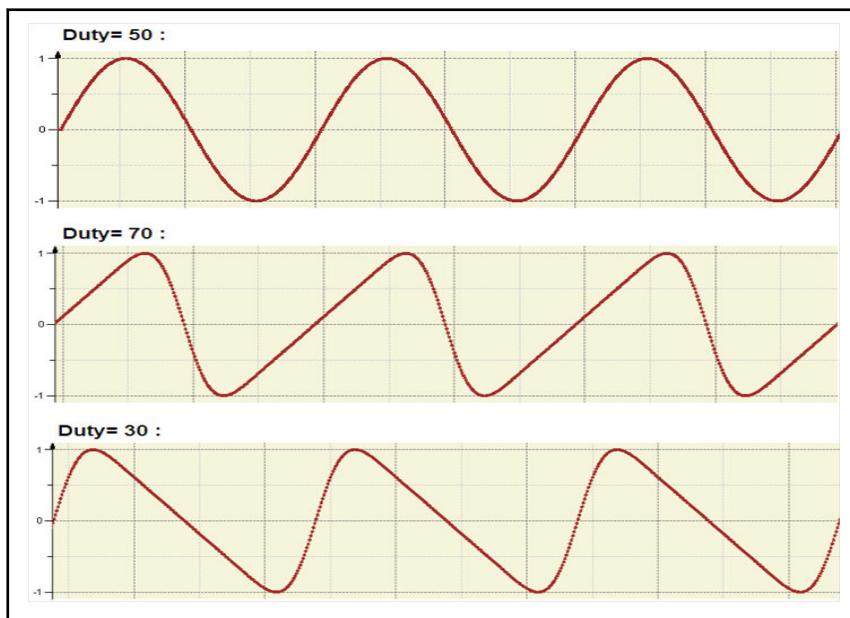


Fig. 7-70: Signal shape: sinusoidal

- DOUBLE_SINE: Double sine

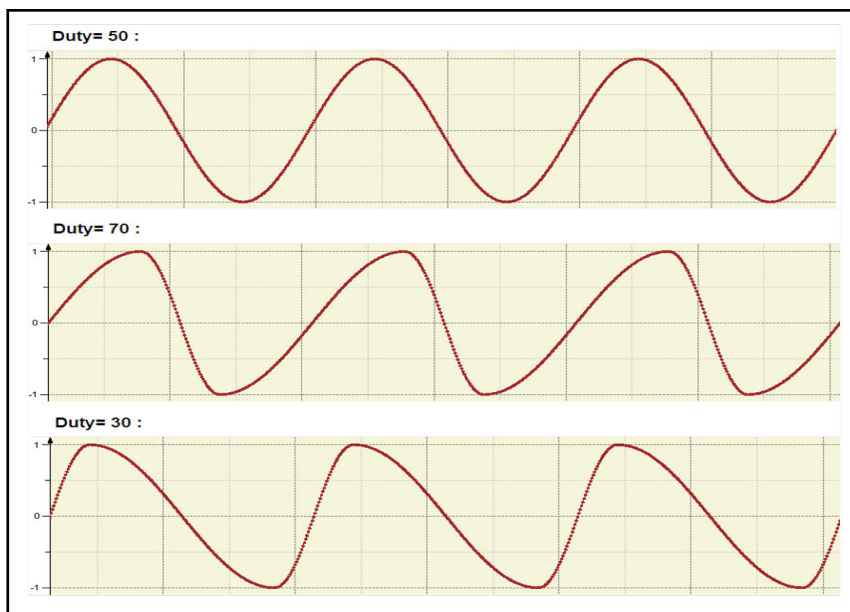


Fig. 7-71: Signal shape: double sine

- TRIANGULAR: Triangle signal

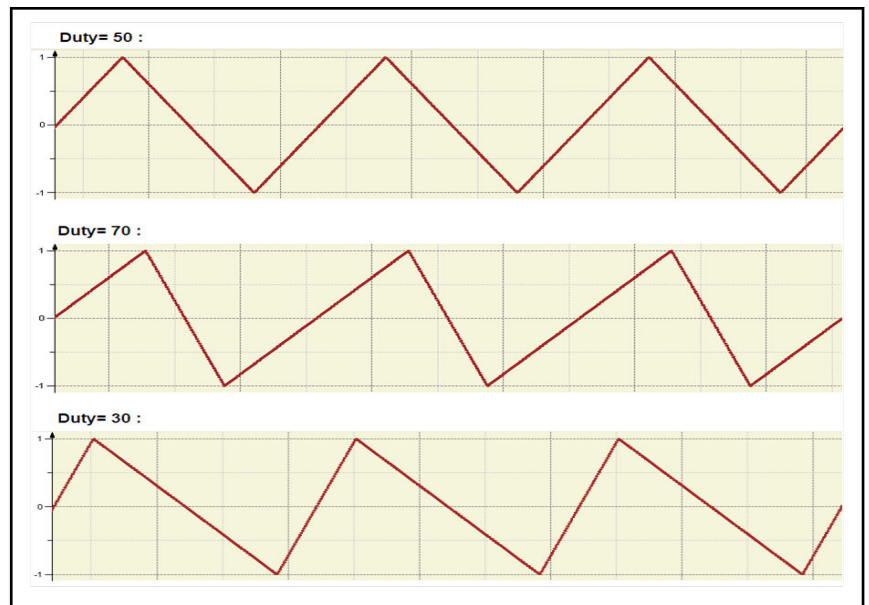


Fig. 7-72: Signal shape: Triangle

Duty, DutyRamp

The "Duty" input determined the sample ratio of the signal. Actually, the 180° mark is moved. With a sample ratio of 50%, the signal is symmetrical. Depending on the signal shape, "Duty" is limited.

The "DutyRamp" input specifies the change rate of the sampling ratio in %/ms if the value is changed at the "Duty" input.

Rounding, RoundingRamp

The "Rounding" input is only enabled if the signal shape "TRIANGULAR" is active. It indicates the rounding of the triangle in %. 0 means no rounding.

The "RoundingRamp" input specifies the change rate of the rounding in %/ms if the "Rounding" input is changed.

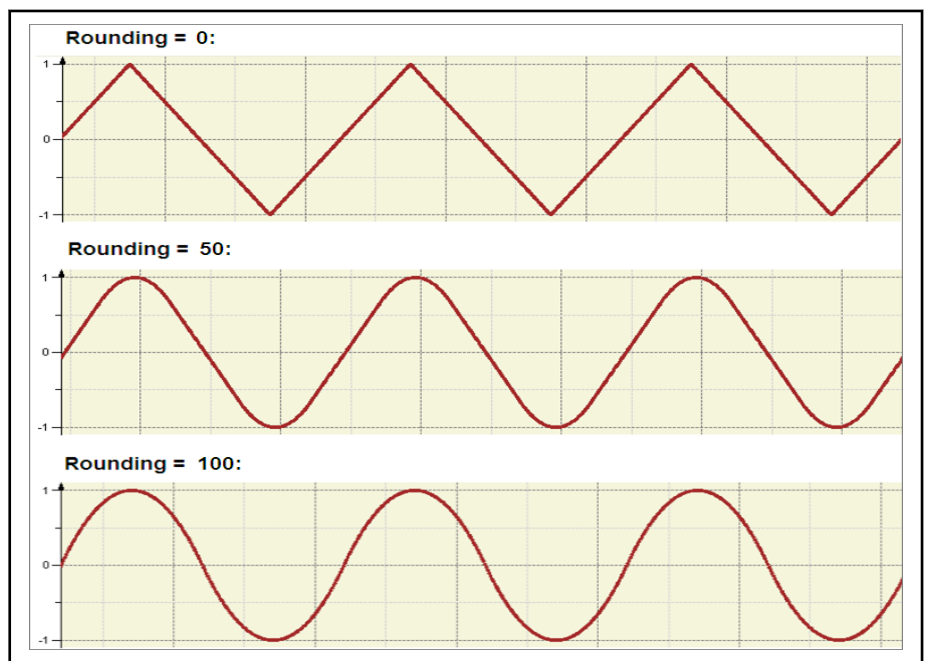


Fig. 7-73: Effects of the "Rounding" input on the signal shape

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000001	"Angle" is lower than 0.0 or higher than 360.0
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000002	"CurveType" is invalid
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000003	"Duty" is outside the permitted range
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000004	"DutyRamp" is smaller than 0.0001
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000005	"Rounding" is negative or higher than 100
INPUT_RANGE_ERROR, 16#0006	16#00000C0F	16#00000006	"RoundingRamp" is smaller than 0.0001
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-119: Generated error numbers of the *IL_SignalGeneratorType02* function block

7.7.11 IL_AmplitudeControllerType01

Brief description This function block controls the amplitude and the zero line of a signal.

Interface description

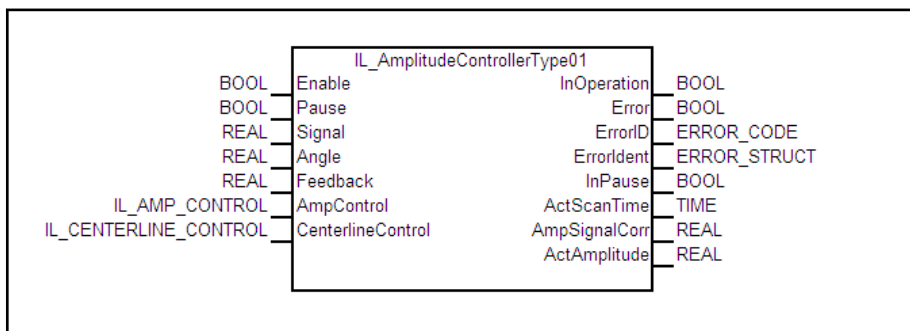


Fig. 7-74: *IL_AmplitudeControllerType01* function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Pauses processing
	Signal	REAL	Input signal
	Angle	REAL	Input angle
	Feedback	REAL	Feedback value for control loop
	AmpControl	IL_AMP_CONTROL	Settings of the amplitude controller IL_AMP_CONTROL , page 428
	Centerline-Control	IL_CENTERLINE_CONTROL	Settings of the zero line controller IL_CENTERLINE_CONTROL , page 428
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes , page 424.
	InPause	BOOL	Pauses active processing

	Name	Type	Comment
	ActScan-Time	TIME	Measured cycle time
	AmpSignal-Corr	REAL	Amplitude-controlled signal
	ActAmplitude	REAL	Latest measured amplitude

Tab. 7-120: IL_AmplitudeControllerType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Signal	REAL	-1.0	1.0	0.0	Continuous
Angle	REAL	0.0	360.0	0.0	Continuous
Feedback	REAL			0.0	Continuous
AmpControl	IL_AMP_CONTROL				Continuous
CenterlineControl	IL_CENTERLINE_CONTROL				Continuous

Tab. 7-121: Min./max. and default values of the IL_AmplitudeControllerType01 inputs

Hidden outputs

To reduce the complexity of the function block, not all internal values are available at the outputs. These values can be directly read at the function block. Write-access is not possible.

Variable	Data type	Description
ActCenterline	REAL	Calculated target zero
ActPhaseShift	REAL	Latest determined phase deviation
AmpCorr	REAL	Correction value of the amplitude
AmpSignal	REAL	Rectified signal without zero line correction
CenterlineCorr	REAL	Controlled zero line
Strobe	BOOL	New cycle detected

Tab. 7-122: Internal values of the function block read via properties

Functional description

The IL_AmplitudeControllerType01 function block controls the amplitude and the zero line of the signal applied at the "Signal" input.

AmpControl

The amplitude is corrected by a PI-controller controlling once per cycle in the zero crossing when the difference between "AmpControl.Target" and the measured amplitude of the "Signal" input is higher than "AmpControl.Window". The correction value is limited to a value between "AmpControl.LowLimit" and "AmpControl.HighLimit".

The control settings "AmpControl.PControl" and "AmpControl.IControl" refer to one cycle. One cycle starts if the "Angle" input jumps from 360 to 0 degrees.

CenterlineControl

The zero crossing correction is corrected by a PI-controller controlling once per cycle in the zero crossing when the difference between "CenterlineControl.Target" and the measured average value of the "Signal" input is higher than "AmpControl.Window". The correction value is limited to a value between "CenterlineControl.LowLimit" and "CenterlineControl.HighLimit".

The control settings "CenterlineControl.PControl" and "CenterlineControl.IControl" refer to one cycle. One cycle starts if the "Angle" input jumps from 360 to 0 degrees.

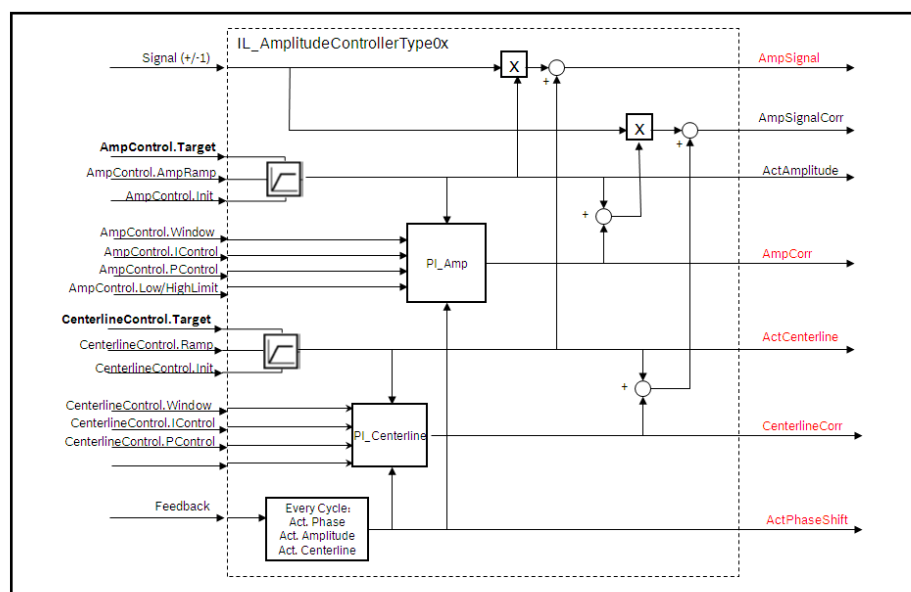


Fig. 7-75: Signal flow of the IL_PhaseController function block

It is processed continuously if the "Enable" input is set.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000001	"Signal" is smaller than -1.0 or higher than 1.0
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000002	"Angle" is smaller than 0° or higher than 360°
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000003	"AmpControl.Target" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000004	"AmpControl.Ramp" is smaller than 0.0001
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000005	"AmpControl.Init" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000006	"AmpControl.Window" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000007	"AmpControl.IControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000008	"AmpControlPControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000009	"AmpControl.LowLimit" is higher than "AmpControl.HighLimit"
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000A	"CenterlineControl.Ramp" is smaller than 0.0001

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000B	"CenterlineControl.Window" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000C	"CenterlineControl.IControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000D	"CenterlineControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000E	"CenterlineControl.LowLimit" is higher than "CenterlineControl.HighLimit"
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-123: Generated error numbers of the IL_AmplitudeControllerType01 function block

7.7.12 IL_AmplitudeControllerType02

Brief description

This function block controls the amplitude and the zero line of a signal.

Interface description

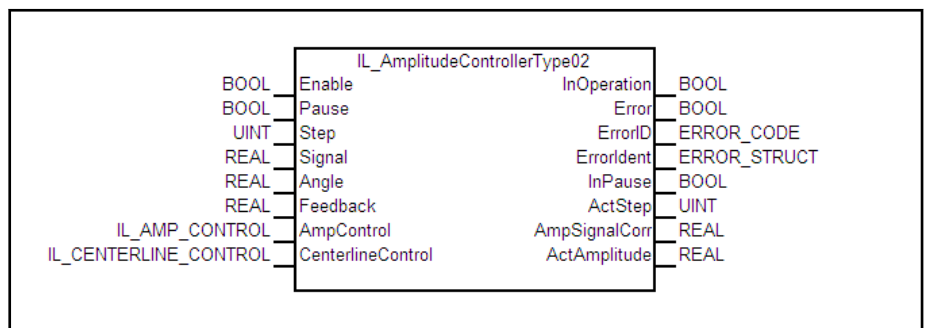


Fig. 7-76: IL_AmplitudeControllerType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Pauses processing
	Step	UINT	Processing step
	Signal	REAL	Input signal
	Angle	REAL	Input angle
	Feedback	REAL	Feedback value for control loop
	AmpControl	IL_AMP_CONTROL	Setting of the amplitude controller IL_AMP_CONTROL , page 428
	Centerline-Control	IL_CENTERLINE_CONTROL	Settings of the zero line controller IL_CENTERLINE_CONTROL , page 428
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error division, see Error codes , page 427
	InPause	BOOL	Pauses active processing
	ActStep	UINT	Latest step processed

	Name	Type	Comment
	AmpSignal-Corr	REAL	Amplitude-controlled signal
	ActAmplitude	REAL	Latest measured amplitude

Tab. 7-124: IL_AmplitudeControllerType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Signal	REAL	-1.0	1.0	0.0	Change at "Step"
Angle	REAL	0.0	360.0	0.0	Change at "Step"
Feedback	REAL			0.0	Change at "Step"
AmpControl	IL_AMP_CONTROL				Change at "Step"
CenterlineControl	IL_CENTERLINE_CONTROL				Change at "Step"

Tab. 7-125: Min./max. and default values of the IL_AmplitudeControllerType02 inputs

Hidden outputs

To reduce the complexity of the function block, not all internal values are available at the outputs. These values can be directly read at the function block. Write-access is not possible.

Variable	Data type	Description
ActCenterline	REAL	Calculated target zero
ActPhaseShift	REAL	Latest determined phase deviation
AmpCorr	REAL	Correction value of the amplitude
AmpSignal	REAL	Rectified signal without zero line correction
CenterlineCorr	REAL	Controlled zero line
Strobe	BOOL	New cycle detected

Tab. 7-126: Internal values of the function block read via properties

Functional description

The IL_AmplitudeControllerType02 function block controls the amplitude and the zero line of the signal applied at the "Signal" input.

AmpControl

The amplitude is corrected by a PI-controller controlling once per cycle in the zero crossing when the difference between "AmpControl.Target" and the measured amplitude of the "Signal" input is higher than "AmpControl.Window". The correction value is limited to a value between "AmpControl.LowLimit" and "AmpControl.HighLimit".

The control settings "AmpControl.PControl" and "AmpControl.IControl" refer to one cycle. One cycle starts if the "Angle" input jumps from 360 to 0 degrees.

CenterlineControl

The zero crossing correction is corrected by a PI-controller controlling once per cycle in the zero crossing when the difference between "CenterlineControl.Target" and the measured average value of the "Signal" input is higher than "AmpControl.Window". The correction value is limited to a value between "CenterlineControl.LowLimit" and "CenterlineControl.HighLimit".

The control settings "CenterlineControl.PControl" and "CenterlineControl.IControl" refer to one cycle. One cycle starts if the "Angle" input jumps from 360 to 0 degrees.

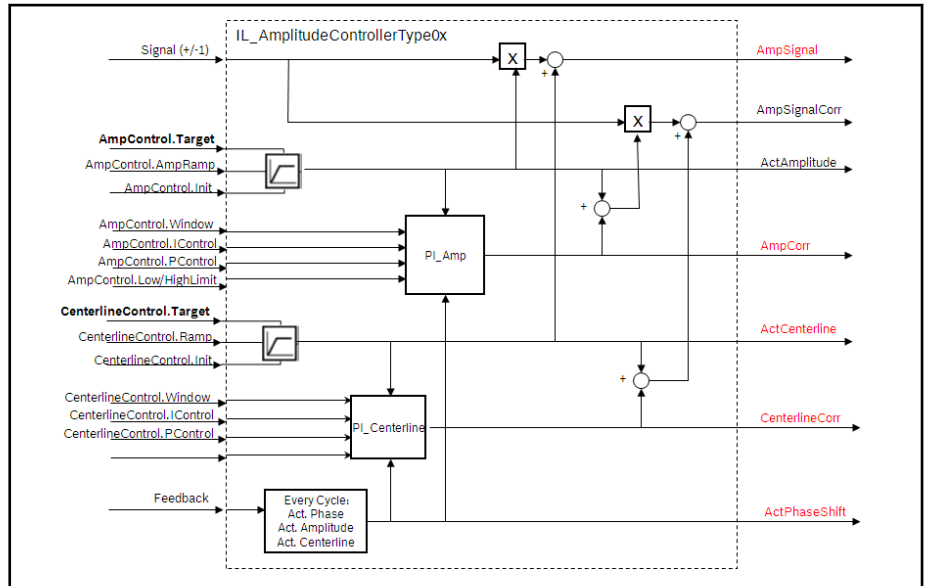


Fig. 7-77: Signal flow of the IL_PhaseController function block

The processing takes place when there is a change at the "Step" input when the "Enable" input is set.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000001	"Signal" is smaller than -1.0 or higher than 1.0
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000002	"Angle" is smaller than 0° or higher than 360°
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000003	"AmpControl.Target" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000004	"AmpControl.Ramp" is smaller than 0.0001
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000005	"AmpControl.Init" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000006	"AmpControl.Window" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000007	"AmpControl.IControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000008	"AmpControl.PControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#00000009	"AmpControl.LowLimit" is higher than "AmpControl.HighLimit"
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000A	"CenterlineControl.Ramp" is smaller than 0.0001
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000B	"CenterlineControl.Window" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000C	"CenterlineControl.IControl" is negative
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000D	"CenterlineControl" is negative

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR, 16#0001	16#00000C0D	16#0000000E	"CenterlineControl.LowLimit" is higher than "CenterlineControl.HighLimit"
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-127: Generated error numbers of the IL_AmplitudeControllerType02 function block

7.7.13 IL_AMP_CONTROL

Brief description The IL_AMP_CONTROL structure contains information required for the amplitude correction.

The amplitude is corrected by a PI-controller controlling once per cycle in the zero crossing when the difference between "AmpControl.Target" and the measured amplitude of the "Signal" input is higher than "AmpControl.Window". The correction value is limited to a value between "AmpControl.LowLimit" and "AmpControl.HighLimit".

Declaration of the IL_AMP_CONTROL structure

```

TYPE IL_AMP_CONTROL:
STRUCT
  Target:    REAL:= 0; // target Amplitude
  Ramp:     REAL:= 0.0001; // Ramp for change amplitude value /s
  Init:     REAL:= 0; // Start Amplitude
  Window:   REAL:= 0; // Window amplitudecorrection (outside this window correction i
  IControl: REAL:= 0; // Integration time Tn for the integration fraction of the cont
  PControl: REAL:= 0; // Proportional value for the gaining fraction of the controller
  HighLimit: REAL:= 1; // Upper limit of amplitudecorrection
  LowLimit: REAL:= 0; // Lower limit of amplitudecorrection
END_STRUCT
END_TYPE

```

Used in:

- [IL_AmplitudeControllerType01, page 422](#)
- [IL_AmplitudeControllerType02, page 425](#)

7.7.14 IL_CENTERLINE_CONTROL

Brief description The IL_CENTERLINE_CONTROL structure contains information required for the zero line correction.

The zero crossing correction is corrected by a PI-controller controlling once per cycle in the zero crossing when the difference between "CenterlineControl.Target" and the measured average value of the "Signal" input is higher than "AmpControl.Window". The correction value is limited to a value between "CenterlineControl.LowLimit" and "CenterlineControl.HighLimit".

The control settings "CenterlineControl.PControl" and "CenterlineControl.IControl" refer to one cycle. One cycle starts if the "Angle" input jumps from 360 to 0 degrees.

Declaration of the IL_CENTERLINE_CONTROL structure

```

TYPE IL_CENTERLINE_CONTROL:
STRUCT
  Target:    REAL:= 0; // Centerline amplitude
  Ramp:     REAL:= 0.0001; // Ramp for change centerline value /s
  Init:     REAL:= 0; // Start centerline
  Window:   REAL:= 0; // Window centerlinecorrection (outside this window correction i
  IControl: REAL:= 0; // Integration time Tn for the integration fraction of the cont
  PControl: REAL:= 0; // Proportional value for the gaining fraction of the controller
  HighLimit: REAL:= 1; // Upper limit of centerlinecorrection
  LowLimit: REAL:= 0; // Lower limit of centerlinecorrection
END_STRUCT
END_TYPE

```

Used in:

- [IL_AmplitudeControllerType01, page 422](#)
- [IL_AmplitudeControllerType02, page 425](#)

7.7.15 IL_CURVETYPE

Brief description The IL_CURVETYPE enumeration type contains the possible curve shapes of the signal generators.

Declaration of the IL_CURVETYPE structure

```
TYPE IL_CURVETYPE :
(
    SINUSOIDAL := 1,
    DOUBLE_SIN := 2,
    TRIANGULAR := 3,
);
END_TYPE
```

SINUSOIDAL: Function similar to sinusoidal with longest consistently increasing and decreasing sections possible

DOUBLE_SINE: Double sine

TRIANGULAR: Triangle signal

Used in:

- [IL_SignalGeneratorType01, page 414](#)
- [IL_SignalGeneratorType02, page 418](#)

7.8 Statistics (statistic function blocks)

7.8.1 Introduction and overview

This section contains the RIL_LoopControl statistic FBs. Statistic FBs evaluate a data signal and generate statistical values.

The RIL_LoopControl provides the following statistic FBs:

Time-discrete	Event-controlled	Description
IL_HistogramType01, page 429	IL_HistogramType02, page 431	Sorts values into categories
IL_StatisticType01, page 433	IL_StatisticType02, page 435	Detects floating mean value, minimum value and maximum value
IL_SuppressionType01, page 438	IL_SuppressionType02, page 442	suppresses measured values with an excessive deviation to the mean value of the measured values (1 limit)
IL_SuppressionType03, page 447	IL_SuppressionType04, page 452	suppresses measured values with an excessive deviation to the mean value of the measured values (2 limits)

Tab. 7-128: RIL_LoopControl statistic function blocks

7.8.2 IL_HistogramType01

Brief description This function block counts the occurrence of values into preset categories of the same size. This block operates continuously.

Interface description

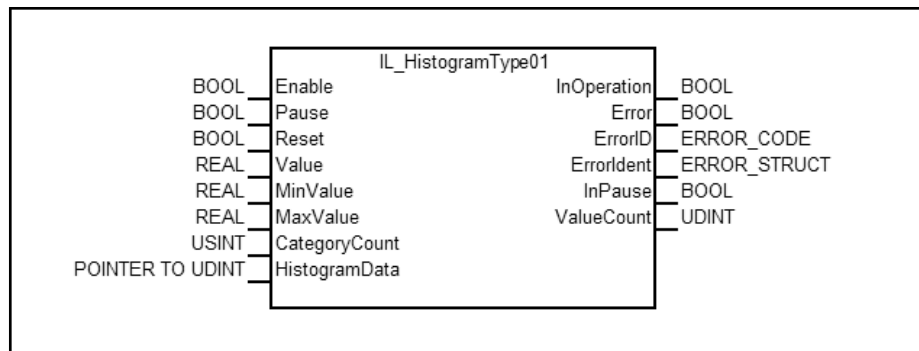


Fig. 7-78: IL_HistogramType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Temporary suspension of processing. The outputs are frozen.
	Reset	BOOL	Reset of the values in "HistogramData" and "ValueCount" when there is a rising edge
	Value	REAL	Value to be split into categories
	MinValue	REAL	Lower limit of classification
	MaxValue	REAL	Upper limit of classification
	Category-Count	USINT	Number of categories into which "Value" should be sub-divided.
	Histogram-Data	POINTER TO UDINT	Address of an array with the values of the histogram categories. The array has to have at least "CategoryCount" elements.
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 431 .
	InPause	BOOL	The evaluation was stopped. The values in "HistogramData" and "ValueCount" do not change.
	ValueCount	UDINT	Number of the values already sorted in "HistogramData".

Tab. 7-129: IL_HistogramType01 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Reset	BOOL			FALSE	Continuous
Value	REAL	n. def	n.def.	0.0	Continuous
MinValue	REAL	n.def.	< MaxValue	0.0	Rising edge at Enable
MaxValue	REAL	> MinValue	n.def.	1.0	Rising edge at Enable

Name	Type	Min. value	Max. value	Default value	Effective
CategoryCount	USINT	1	255	10	Rising edge at Enable
HistogramData	POINTER TO UDINT	0	n.def.	0	Rising edge at Enable

Tab. 7-130: Min./max. and default values of the IL_HistogramType01 inputs

Functional description

The IL_HistogramType01 function block sorts the values at "Value" between "MinValue" and "MaxValue" into one of the "CategoryCount" categories. Each category is the same size and covers a specific non-overlapping interval. The interval range is specifically defined by $(\text{"MaxValue"} - \text{"MinValue"}) / \text{"CategoryCount"}$. Each category counts the "Value" values falling into it. The total number of values already applied, is issued in "ValueCount".

"HistogramData" points to an array of UDINTs. The array has to have at least UDINT-type "CategoryCount" elements. Each element of the array forms a category and counts the occurrence of "Value" within the category.

It is processed continuously if the "Enable" input is set.

Pause, InPause

If "Pause" is set, the current state is frozen. "InPause" is set and no further evaluation of "Value" takes place.

Reset

If there is a rising edge at "Reset", all the elements of the array at "HistogramData" are set to 0 and the "ValueCount" output is deleted.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C07	16#00000001	"MinValue" is greater than "MaxValue".
INPUT_RANGE_ERROR, 16#0006	16#00000C07	16#00000002	"CategoryCount" is 0
INPUT_INVALID_ERROR, 16#0001	16#00000C07	16#00000003	"HistogramData" is 0
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-131: Generated error numbers of the IL_HistogramType0 function block

7.8.3 IL_HistogramType02

Brief description

This function block counts the occurrence of values into preset categories of the same size. This block only operates when there is a change at the "Step" input.

Interface description

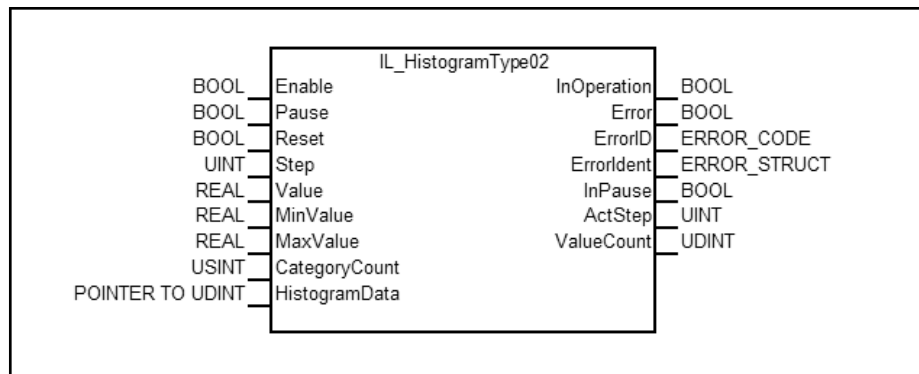


Fig. 7-79: IL_HistogramType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Processing of function block enabled
	Pause	BOOL	Temporary suspension of processing. The outputs are frozen.
	Reset	BOOL	Reset of the values in "HistogramData" and "ValueCount" when there is a rising edge
	Step	UINT	Current processing step, a new value at "Value" is only processed when there is a change at "Step"
	Value	REAL	Value to be split into categories
	MinValue	REAL	Lower limit of classification
	MaxValue	REAL	Upper limit of classification
	Category-Count	USINT	Number of categories into which "Value" should be sub-divided.
	Histogram-Data	POINTER TO UDINT	Address of an array with the values of the histogram categories. The array has to have at least "CategoryCount" elements.
VAR_OUTPUT	InOperation	BOOL	The function block operates. The outputs are valid.
	Error	BOOL	Indicates that an error occurred in the function block.
	ErrorID	ERROR_CODE	Error ID
	ErrorIdent	ERROR_STRUCT	For the error structure with further error subdivision, see Error codes, page 433 .
	InPause	BOOL	The evaluation was stopped. The values in "HistogramData" and "ValueCount" do not change.
	ActStep	UINT	Last processed step
	ValueCount	UDINT	Number of the values already sorted in "HistogramData".

Tab. 7-132: IL_HistogramType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Reset	BOOL			FALSE	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
Step	UINT	0	65335	0	Continuous
Value	REAL	n. def	n.def.	0.0	If "Step" is changing
MinValue	REAL	n.def.	< MaxValue	0.0	Rising edge at Enable
MaxValue	REAL	> MinValue	n.def.	1.0	Rising edge at Enable
CategoryCount	USINT	1	255	10	Rising edge at Enable
HistogramData	POINTER TO UDINT	0	n.def.	0	Rising edge at Enable

Tab. 7-133: Min./max. and default values of the IL_HistogramType02 inputs

Functional description

The IL_HistogramType02 function block sorts the values at "Value" between "MinValue" and "MaxValue" into one of the "CategoryCount" categories. Each category is the same size and covers a specific non-overlapping interval. The interval range is specifically defined by $(\text{"MaxValue"} - \text{"MinValue"}) / \text{"CategoryCount"}$. Each category counts the "Value" values falling into it. The total number of values already applied, is issued in "ValueCount". The evaluation of "Value" only takes place if the "Step" input has changed.

"HistogramData" points to an array of UDINTs. The array has to have at least UDINT-type "CategoryCount" elements. Each element of the array forms a category and counts the occurrence of "Value" within the category.

Processing only takes place if the value at the "Step" input changes.

Pause, InPause

If "Pause" is set, the current state is frozen. "InPause" is set and no further evaluation of "Value" takes place.

Reset

If there is a rising edge at "Reset", all the elements of the array at "HistogramData" are set to 0 and the "ValueCount" output is deleted.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR, 16#0001	16#00000C07	16#00000001	"MinValue" is greater than "MaxValue".
INPUT_RANGE_ERROR, 16#0006	16#00000C07	16#00000002	"CategoryCount" is 0
INPUT_INVALID_ERROR, 16#0001	16#00000C07	16#00000003	"HistogramData" is 0
STATE_MACHINE_ERROR, 16#0005	16#00000006	16#00000000	Runtime error in function block: Invalid state of the state machine.

Tab. 7-134: Generated error numbers of the IL_HistogramType02 function block

7.8.4 IL_StatisticType01

Brief description

The FB identifies time-discrete statistical characteristics of measured values or control deviations.

Characteristics identified:

- Mean value (exponentially floating average value)
- Variance (exponentially floating variance)
- Minimum/Maximum values (exponentially falling, that is the values age)

Interface description

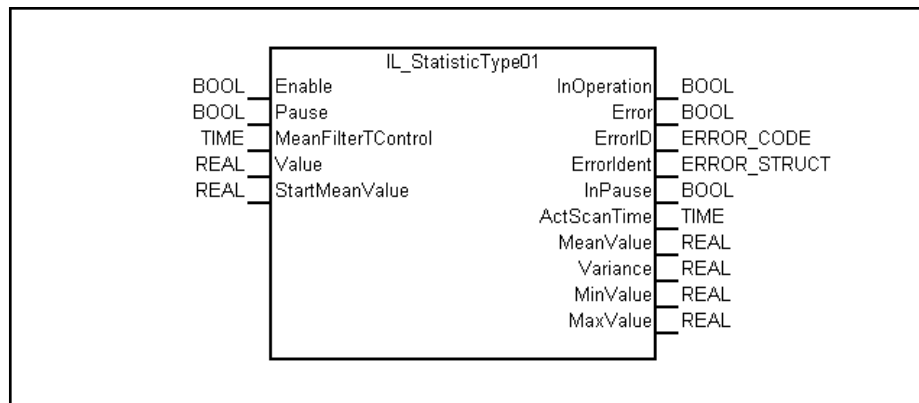


Fig. 7-80: IL_StatisticType01 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Calculation of data in each cycle when "Enable" = TRUE.
	Pause	BOOL	The output data is not changed when Pause = TRUE.
	MeanFilterTControl	TIME	The constant "Tn" of the exponentially floating calculation.
	Value	REAL	Input value, the statistical parameters of which are identified.
	StartMeanValue	REAL	The startup value, which is applied when there is a rising edge at "Enable" at the "MeanValue" output.
VAR_OUTPUT	InOperation	BOOL	Processing is running, outputs are valid.
	Error	BOOL	Error (see ErrorID and ErrorIdent)
	ErrorID	ERROR_CODE	Error description
	ErrorIdent	ERROR_STRUCT	For a detailed error description, see Error codes, page 435 .
	InPause	BOOL	Indicates that the "Pause" input is active and the values in "MeanValue", "Variance", "MinValue" and "MaxValue" do not change.
	ActScanTime	TIME	The time difference since the function block was called last time.
	MeanValue	REAL	Mean value of "Value" (exponentially floating calculation)
	Variance	REAL	Variance of "Value" (exponentially floating calculation)
	MinValue	REAL	Minimum value of "Value" (exponentially floating calculation)
	MaxValue	REAL	Maximum value of "Value" (exponentially floating calculation)

Tab. 7-135: IL_StatisticType01 function block

Min./max. and default values of
the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
MeanFilterTControl	TIME	(2 x ActScan-Time)	n. def	100 ms	Continuous
Value	REAL	n. def	n. def	0.0	Continuous
StartMeanValue	REAL	n. def	n. def	0.0	Rising edge at Enable

Tab. 7-136: Min./max. and default values of the IL_StatisticType01 inputs

Functional description

The function block identifies the statistical parameters of the "Value" input value. The parameters can, for example, be used to show the mean value, variance and minimum/maximum values during the control processes.

"MeanValue" and "Variance"

The calculation takes into account older input values to a lesser extent than newer input values (so-called exponentially floating calculation). This ensures that the characteristics do not fully contain all measured values since activation of the function block, but also take into account changes to the input parameters. For example, a step change of the mean value of the "Value" input variable to a new mean value of said variable, is relatively quickly displayed at the "MeanValue" output, without older input values distorting the new mean value. Calculation of the variance takes place in the same way.

The averaging is affected by the "MeanFilterTControl" input. Larger values take into account older data to a greater extent to determine the statistical characteristic variables. The higher the value, the slower the outputs react to changes to the characteristic variables of the input signal and the more constant are the characteristic values identified.

"MinValue" and "MaxValue"

The outputs show the tendency of the minimum and maximum values that occur. As with the mean value and the variance, not all input values from activation of the function block are evaluated, rather the minimum and maximum values are identified in a floating manner. This means that a minimum or maximum value that occurs ages and is over-written by newer (possibly lower) values. The function is comparable with a following number, which is immediately taken with a new higher number but then falls again with a large time constant.

The "MeanFilterTControl" input also shows the time-lag of the output in relation to the lowering of the "MinValue" and "MaxValue" values.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state
INPUT_RANGE_ERROR	16#0C02	16#0005	Value of "MeanFilterTControl" < (2 x "ActScan-Time")

Tab. 7-137: Generated error numbers of the IL_StatisticType01 function block

7.8.5 IL_StatisticType02

Brief description

The FB identifies event-controlled statistical characteristics of measured values or control deviations.

Characteristics identified:

- Mean value (exponentially floating average value)
- Variance (exponentially floating variance)
- Minimum/Maximum values (exponentially falling, that is the values age)

Interface description

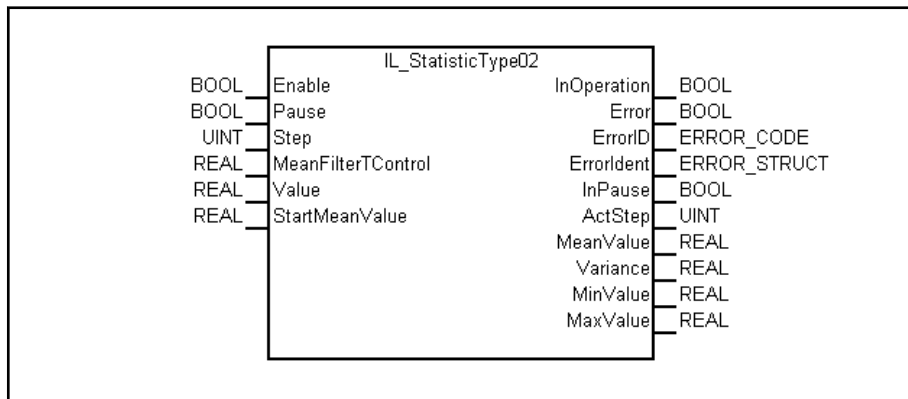


Fig. 7-81: IL_StatisticType02 function block

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Calculation of the data each time "Step" changes, providing Enable = TRUE
	Pause	BOOL	The OutputValue data is not changed, providing "Pause" = TRUE. If the "Step" input increases in the meantime, the step that is currently at the "ActStep" output is issued.
	Step	UINT	Current calculation step
	MeanFilterTControl	REAL	The constant "Tn" of the exponentially floating calculation relating to "Step".
	Value	REAL	Input value, the statistical parameters of which are identified.
	StartMeanValue	REAL	The startup value, which is applied when there is a rising edge at "Enable" at the "MeanValue" output.
VAR_OUTPUT	InOperation	BOOL	Processing without errors, output data is valid.
	Error	BOOL	Error (see ErrorID and ErrorIdent)
	ErrorID	ERROR_CODE	Error description
	ErrorIdent	ERROR_STRUCT	For a detailed error description, see Error codes, page 437 .
	InPause	BOOL	Indicates that the "Pause" input is active and the values in "MeanValue", "Variance", "MinValue" and "MaxValue" do not change.
	ActStep	UINT	"Step" value to which "OutputValue" refers. "ActStep" is initialized with the value of "Step" when there is a rising edge at "Enable".
	MeanValue	REAL	Mean value of "Value" (exponentially floating calculation)
	Variance	REAL	Variance of "Value" (exponentially floating calculation)

	Name	Type	Comment
	MinValue	REAL	Minimum value of "Value" (exponentially floating calculation)
	MaxValue	REAL	Maximum value of "Value" (exponentially floating calculation)

Tab. 7-138: IL_StatisticType02 function block

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	16#FFFF	0	Continuous
MeanFilterTControl	REAL	2.0	n.def.	100.0	Continuous
Value	REAL	n.def.	n.def.	0.0	Continuous
StartMeanValue	REAL	n.def.	n.def.	0.0	Rising edge at Enable

Tab. 7-139: Min./max. and default values of the IL_StatisticType02 inputs

Functional description

The function block identifies the statistical parameters of the "Value" input value. The parameters can, for example, be used to show the mean value, variance and minimum/maximum values during the control processes.

"MeanValue" and "Variance"

The calculation takes into account older input values to a lesser extent than newer input values (so-called exponentially floating calculation). This ensures that the characteristics do not fully contain all measured values since activation of the function block, but also take into account changes to the input parameters. For example, a step change of the mean value of the "Value" input variable to a new mean value of said variable, is relatively quickly displayed at the "MeanValue" output, without older input values distorting the new mean value. Calculation of the variance takes place in the same way.

The averaging is affected by the "MeanFilterTControl" input. Larger values take into account older data to a greater extent to determine the statistical characteristic variables. The higher the value, the slower the outputs react to changes to the characteristic variables of the input signal and the more constant are the characteristic values identified.

"MinValue" and "MaxValue"

The outputs show the tendency of the minimum and maximum values that occur. As with the mean value and the variance, not all input values from activation of the function block are evaluated, rather the minimum and maximum values are identified in a floating manner. This means that a minimum or maximum value that occurs ages and is over-written by newer (possibly lower) values. The function is comparable with a following number, which is immediately taken with a new higher number but then falls again with a large time constant.

The "MeanFilterTControl" input also shows the time-lag of the output in relation to the lowering of the "MinValue" and "MaxValue" values.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state
INPUT_RANGE_ERROR	16#0C02	16#0005	Value of "MeanFilterTControl" has to be greater or equal to 2.0.

Tab. 7-140: Generated error numbers of the IL_StatisticType02 function block

7.8.6 IL_SuppressionType01

Brief description This time-discreet FB suppresses measured values with an excessive deviation to the mean value of the measured values. The measured values outside the threshold values that can be parameterized are not transferred, but output as filtered value.

Interface description

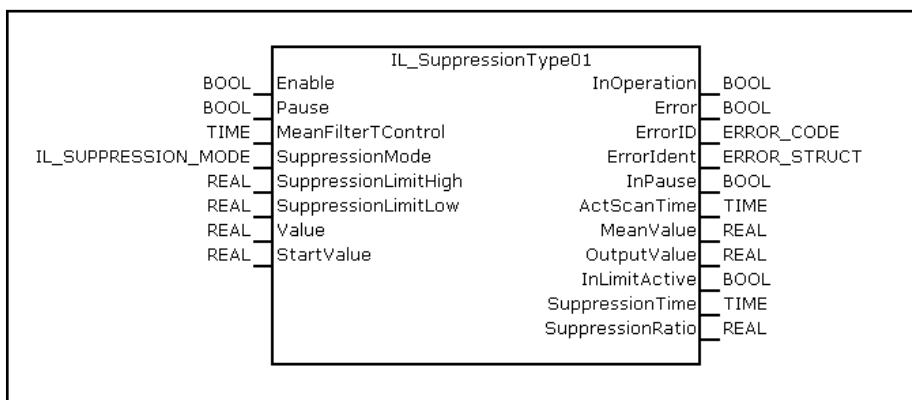


Fig. 7-82: IL_SuppressionType01 FB

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Calculation of data in each cycle while "Enable" = TRUE
	Pause	BOOL	The output data is not changed while "Pause" = TRUE
	MeanFilterTControl	TIME	Constant "Tn" of the (PT1-) mean value filter.
	SuppressionMode	IL_SUPPRESSION_MODE	Describes the function of the inputs "SuppressionLimitLow" and "SuppressionLimitHigh". IL_SUPPRESSION_MODE, page 492
	SuppressionLimitHigh	REAL	Upper value of the suppression window (is added to the mean value).
	SuppressionLimitLow	REAL	Lower value of the suppression window (is subtracted from the mean value).
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
VAR_OUTPUT	InOperation	BOOL	Processing without errors, output data is valid.
	Error	BOOL	Error (see ErrorID and ErrorIdent)
	ErrorID	ERROR_CODE	Error description
	ErrorIdent	ERROR_STRUCT	For a detailed error description, see Error codes, page 442 .

	Name	Type	Comment
	InPause	BOOL	Indicates that the "Pause" input is enabled and the output values do not change.
	ActScanTime	TIME	The time difference since the function block was called last time.
	MeanValue	REAL	Filtered mean value compared to the "Value".
	OutputValue	REAL	The calculated output value.
	InLimitActive	BOOL	Indicates that the suppression window is active.
	SuppressionTime	TIME	Time counter, counts the time the measured values are suppressed. The value is deleted at the rising edge at "Enable" (T#0s) and counts up in case of an active suppression. As soon as an input value, which is not suppressed, is available again, the output is deleted again. In case of an overflow, the counter automatically becomes "0".
	SuppressionRatio	REAL	Percentage rate of the suppressed values (0-100%).

Tab. 7-141: IL_SuppressionType01 FB

IL_SUPPRESSION_MODE is an enumeration type and describes the function of the inputs "SuppressionLimitLow" and "SuppressionLimitHigh" with the following allocation:

Element	Description
IL_SUPPRESSION_PERCENTAGE	Values are on a percentage rate to the averaged input value
IL_SUPPRESSION_ABSOLUTE_VALUE	Values are absolute values to the averaged input value

Tab. 7-142: Structure of the enumeration type IL_SUPPRESSION_MODE

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
MeanFilterTControl	TIME	0.0	n. def	100 ms	Continuous
SuppressionMode	IL_SUPPRESSION_MODE			IL_SUPPRESSION_PERCENTAGE	Rising edge at Enable
SuppressionLimitHigh	REAL	0.0	n. def	20.0	Continuous
SuppressionLimitLow	REAL	0.0	n. def	20.0	Continuous
Value	REAL	n. def	n. def	0.0	Continuous
StartValue	REAL	n. def	n. def	0.0	Rising edge at Enable

Tab. 7-143: Min./max. and default values of the IL_SuppressionType01 inputs

Functional description

When controlling, it is often required that the measured values with an obviously strong deviation from the expected value are filtered (suppressed). Therefore, a suppression window is implemented in the FB that moves along with the mean value of the value to be filtered and forms a tolerance window around this mean value. Each measured value "Value" is compared to the filtered measured value (filtering of the measured value with a PT1 filter). If the respective measured value is not within the suppression window formed around the mean value (tolerance window defined by "SuppressionLimitLow"

or "SuppressionLimitHigh"), it is not output at the the FB output "OutputValue", but suppressed. That means that the mean value is output instead of "Value" at the "OutputValue" output.

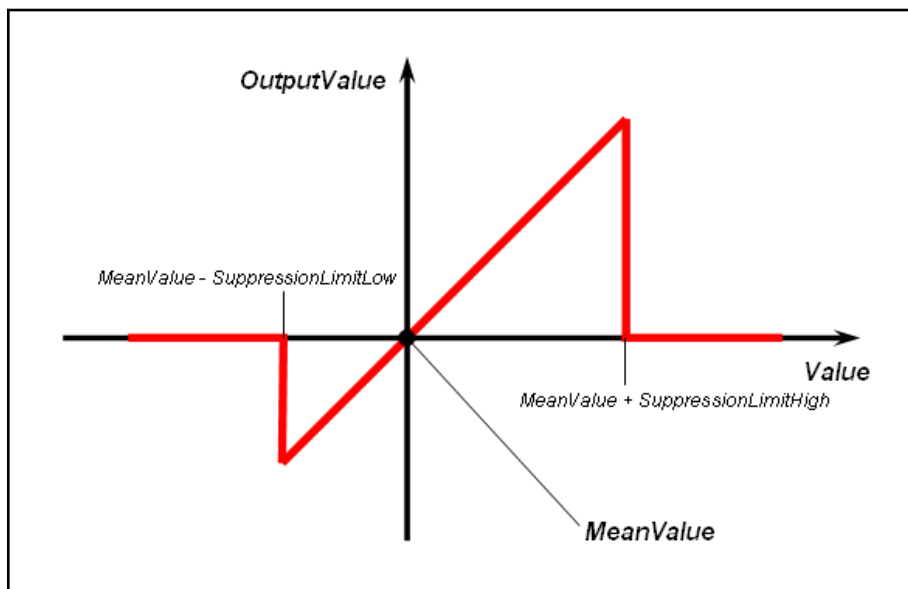


Fig. 7-83: Principle of the suppression window

Internal FB structure

The FB consists of an mean value filter and a block executing the suppression of the measured values. The mean value filtering is executed by a PT1 element. The filter time constant should be higher than the process time constant so that the mean value filtering can act correctly. The filtered mean value can be diagnosed via the "MeanValue" output.

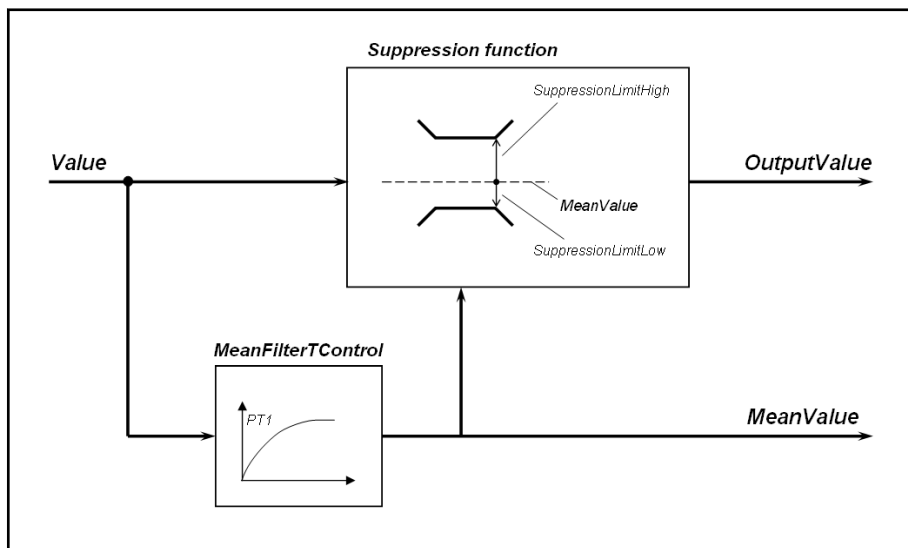


Fig. 7-84: Internal structure of the IL_SuppressionType01

Window-cutting mode

In case of window-cutting, the upper threshold ("SuppressionLimitHigh") and the lower threshold ("SuppressionLimitLow") can be configured separately. Two modes are available:

- **In percentage related to the mean value (IL_SUPPRESSION_PERCENTAGE):** the window width is specified by the percentage rate of the

mean value. That means that "SuppressionLimitHigh" specifies a percentage rate (from the absolute value of the mean value) that is added to the mean value to maintain the upper threshold. "SuppressionLimitLow" specifies a percentage rate (from the absolute value of the mean value) that is subtracted from the mean value for the lower threshold. It is to be considered that the window size might become very small if the amount of the mean value becomes very small.

- **Absolute window related to the mean value (IL_SUPPRESSION_ABSOLUTE_VALUE):** The values at the inputs "SuppressionLimitLow" and "SuppressionLimitHigh" directly specify the window size around the mean value.

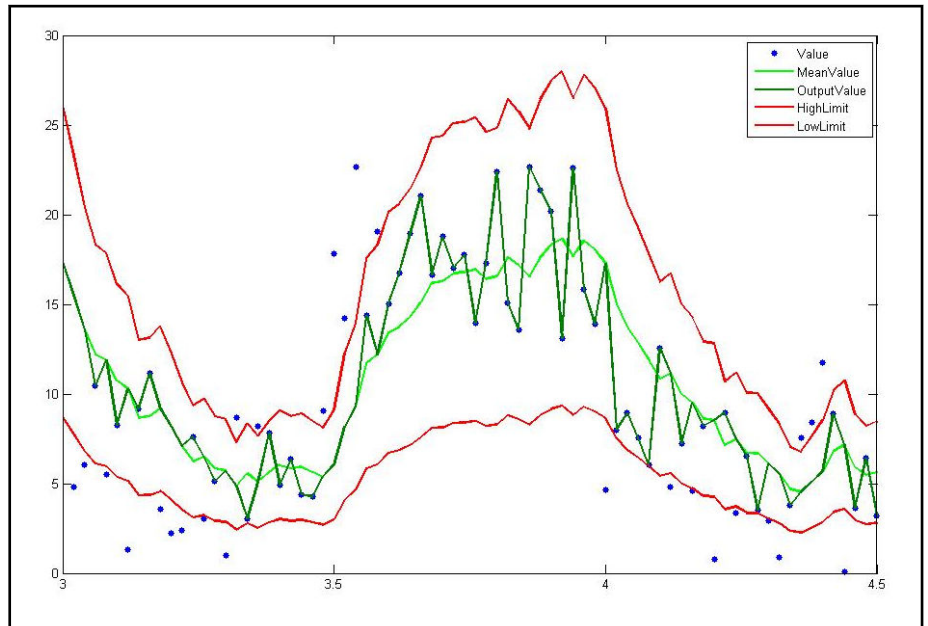


Fig. 7-85: Example of a window-cutting "related to the mean value in percentage"

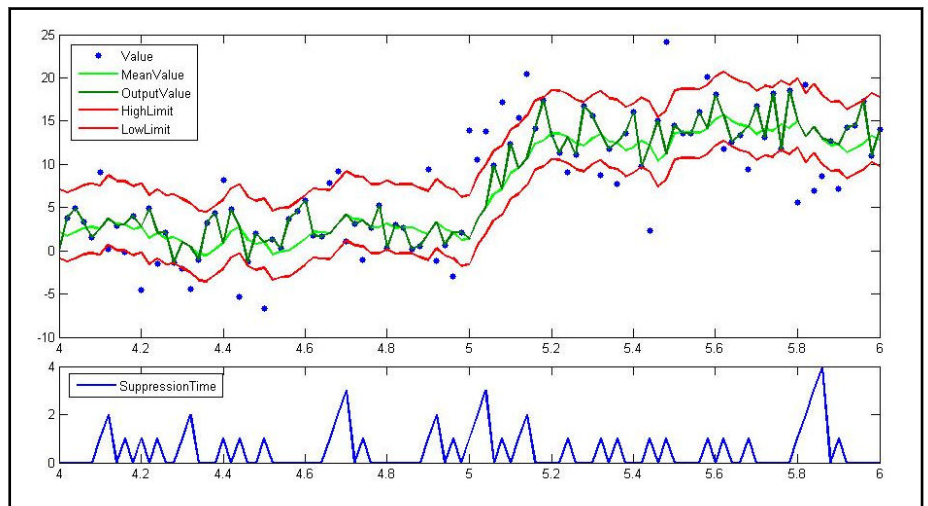


Fig. 7-86: Example of a window-cutting "with absolute window around the mean value"

Diagnostics of the window-cutting

The window-cutting function can be observed for diagnostic purposes via the outputs "SuppressionTime" and "SuppressionRatio". "SuppressionTime" is a time counter that counts the suppression period of the measured values. It

counts up in case of an active suppression and is reset to "0" if an input value that is not suppressed is available. It is also reset to "0" in case of a rising edge at "Enable" as well as in case of an overflow. "SuppressionRatio" is the percentage rate of time of all the values suppressed.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state
INPUT_INVALID_ERROR	16#0C09	16#0002	"SuppressionLimitHigh" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0003	"SuppressionLimitLow" value is lower than 0.0

Tab. 7-144: Generated error numbers of the IL_SuppressionType01 function block

7.8.7 IL_SuppressionType02

Brief description This event-controlled FB suppresses measured values with an excessive deviation to the mean value of the measured values. The measured values outside the threshold values that can be parameterized are not transferred, but output as filtered value.

Interface description

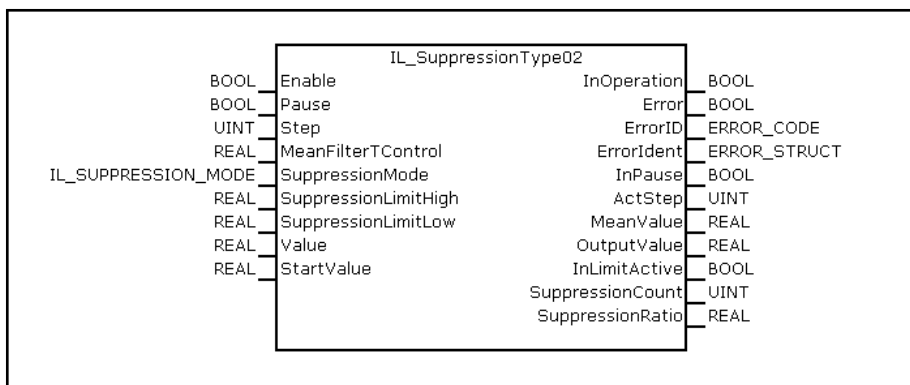


Fig. 7-87: IL_SuppressionType02 FB

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Calculation of data in each cycle (step) while "Enable" = TRUE
	Pause	BOOL	The output data is not changed while "Pause" = TRUE. If the "Step" input increases in the meantime, the step that is currently at the "ActSteps" output is issued.
	Step	UINT	Current calculation step.
	MeanFilterTControl	REAL	The constant "Tn" of the mean value filter element related to "Step".
	SuppressionMode	IL_SUPPRESSION_MODE	Describes the function of the inputs "SuppressionLimitLow" and "SuppressionLimitHigh". IL_SUPPRESSION_MODE, page 492
	SuppressionLimitHigh	REAL	Upper value of the suppression window (is added to the mean value).
	SuppressionLimitLow	REAL	Lower value of the suppression window (is subtracted from the mean value).

	Name	Type	Comment
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
VAR_OUTPUT	InOperation	BOOL	Processing without errors, output data is valid.
	Error	BOOL	Error (see ErrorID and ErrorIdent)
	ErrorID	ERROR_CODE	Error description
	ErrorIdent	ERROR_STRUCT	For a detailed error description, see Error codes, page 446 .
	InPause	BOOL	Indicates that the "Pause" input is enabled and the output values do not change.
	ActStep	UINT	Step value to which "OutputValue" refers. "ActStep" is initialized with the value of "Step" at a rising edge at "Enable".
	MeanValue	REAL	Filtered mean value compared to the "Value".
	OutputValue	REAL	The calculated output value.
	InLimitActive	BOOL	Indicates that the suppression window is active.
	SuppressionCount	UINT	Counter for the number of suppressed values. The value is deleted at the rising edge at "Enable" and counts up in case of an active suppression. As soon as an input value, which is not suppressed, is available again, the output is deleted again. In case of an overflow (16xFFFF), the counter automatically becomes "0".
	SuppressionRatio	REAL	Percentage rate of the suppressed values (0-100%).

Tab. 7-145: IL_SuppressionType02 FB

IL_SUPPRESSION_MODE is an enumeration type and describes the function of the inputs "SuppressionLimitLow" and "SuppressionLimitHigh" with the following allocation:

Element	Description
IL_SUPPRESSION_PERCENTAGE	Values are on a percentage rate to the averaged input value
IL_SUPPRESSION_ABSOLUTE_VALUE	Values are absolute values to the averaged input value

Tab. 7-146: Structure of the enumeration type IL_SUPPRESSION_MODE

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	16#FFFF	0	Continuous
MeanFilterTControl	TIME	0.0	n. def	100 ms	Continuous
SuppressionMode	IL_SUPPRESSION_MODE			IL_SUPPRESSION_PERCENTAGE	Rising edge at Enable
SuppressionLimitHigh	REAL	0.0	n. def	20.0	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
SuppressionLimitLow	REAL	0.0	n. def	20.0	Continuous
Value	REAL	n. def	n. def	0.0	Continuous
StartValue	REAL	n. def	n. def	0.0	Rising edge at Enable

Tab. 7-147: Min./max. and default values of the IL_SuppressionType02 inputs

Functional description

When controlling, it is often required that the measured values with an obviously strong deviation from the expected value are filtered (suppressed). Therefore, a suppression window is implemented in the FB that moves along with the mean value of the value to be filtered and forms a tolerance window around this mean value. Each measured value "Value" is compared to the filtered measured value (filtering of the measured value with a PT1 filter). If the respective measured value is not within the suppression window formed around the mean value (tolerance window defined by "SuppressionLimitLow" or "SuppressionLimitHigh"), it is not output at the the FB output "OutputValue", but suppressed. That means that the mean value is output instead of "Value" at the "OutputValue" output.

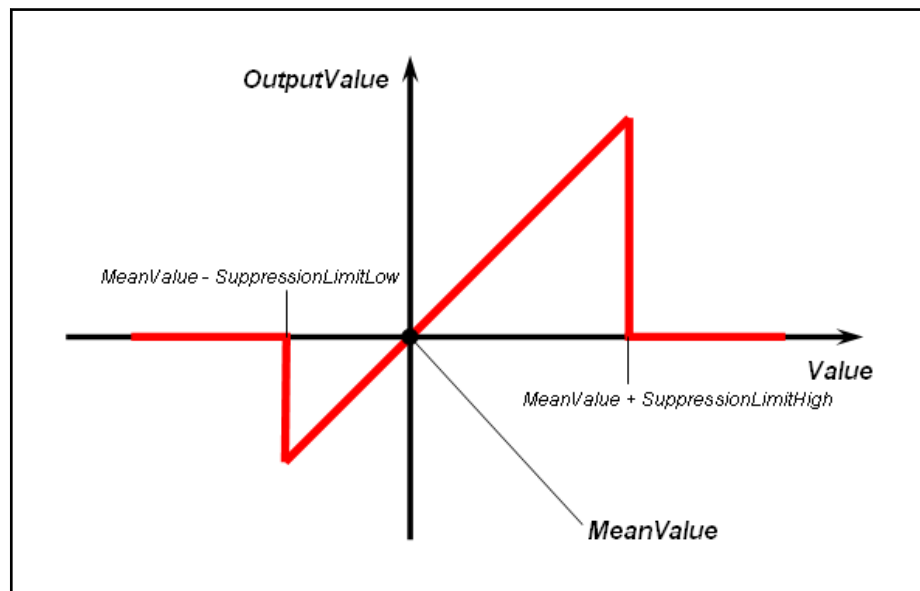


Fig. 7-88: Principle of the suppression window

Internal FB structure

The FB consists of an mean value filter and a block executing the suppression of the measured values. The mean value filtering is executed by a PT1 element. The filter time constant should be higher than the process time constant so that the mean value filtering can act correctly. The filtered mean value can be diagnosed via the "MeanValue" output.

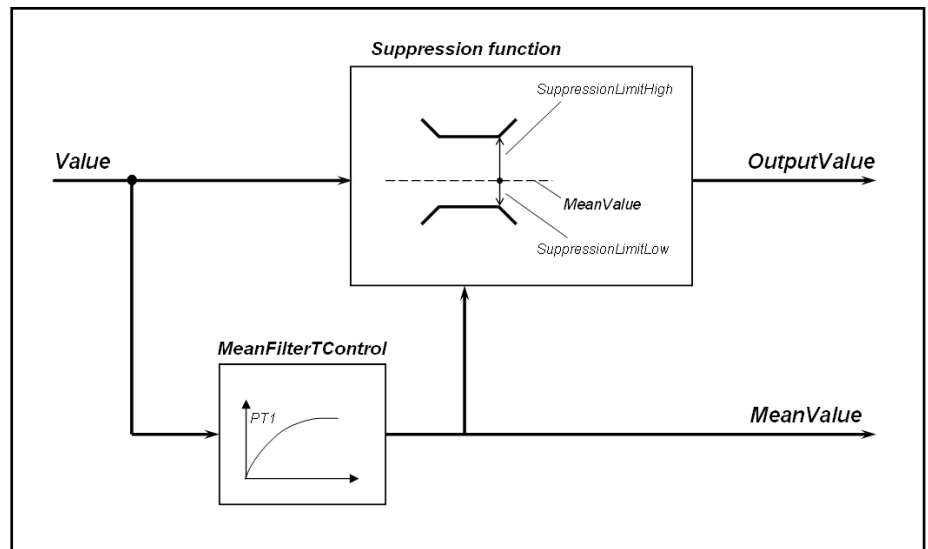


Fig. 7-89: Internal structure of the `IL_SuppressionType02`

Window-cutting mode

In case of window-cutting, the upper threshold ("`SuppressionLimitHigh`") and the lower threshold ("`SuppressionLimitLow`") can be configured separately. Two modes are available:

- **In percentage related to the mean value (`IL_SUPPRESSION_PERCENTAGE`):** the window width is specified by the percentage rate of the mean value. That means that "`SuppressionLimitHigh`" specifies a percentage rate (from the absolute value of the mean value) that is added to the mean value to maintain the upper threshold. "`SuppressionLimitLow`" specifies a percentage rate (from the absolute value of the mean value) that is subtracted from the mean value for the lower threshold. It is to be considered that the window size might become very small if the amount of the mean value becomes very small.
- **Absolute window related to the mean value (`IL_SUPPRESSION_ABSOLUTE_VALUE`):** The values at the inputs "`SuppressionLimitLow`" and "`SuppressionLimitHigh`" directly specify the window size around the mean value.

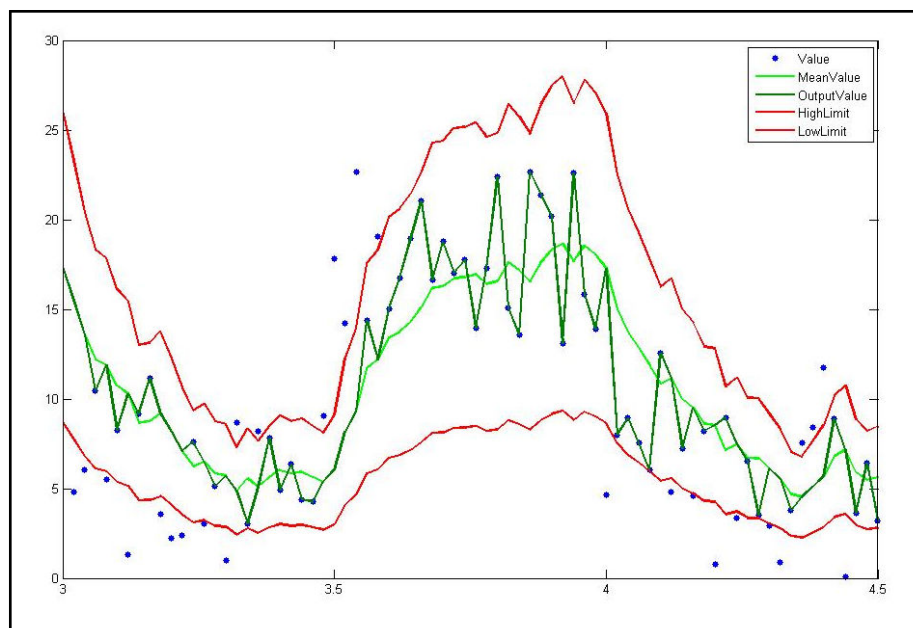


Fig. 7-90: Example of a window-cutting "related to the mean value in percentage"

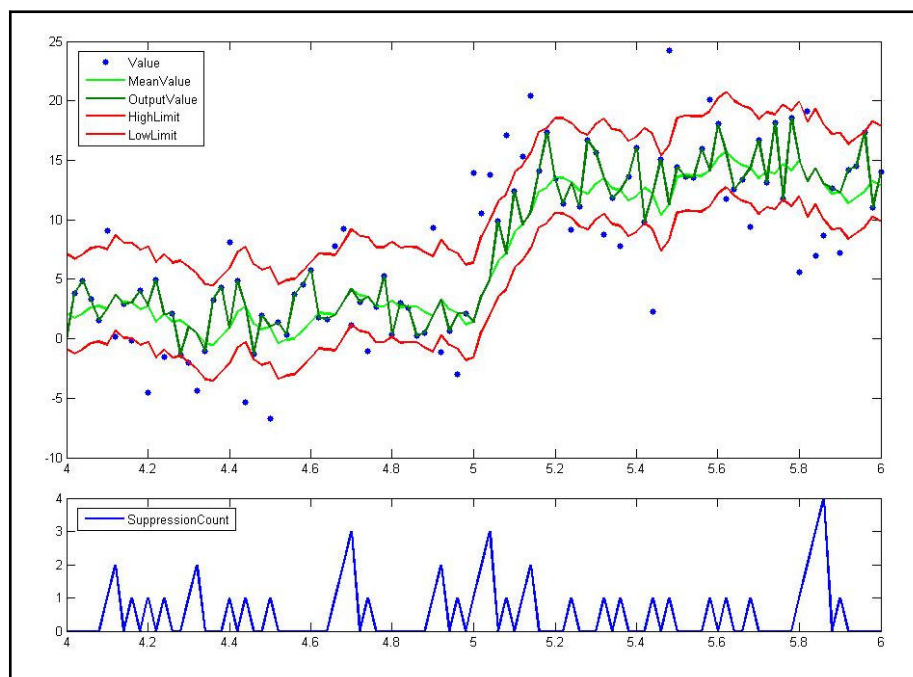


Fig. 7-91: Example of a window-cutting "with absolute window around the mean value"

Diagnostics of the window-cutting

The window-cutting function can be observed for diagnostic purposes via the outputs "SuppressionCount" and "SuppressionRatio". "SuppressionCount" is a counter that counts the suppression period of the measured values. It counts up in case of an active suppression and is reset to "0" if an input value that is not suppressed is available. It is also reset to "0" in case of a rising edge at "Enable" as well as in case of an overflow. "SuppressionRatio" is the percentage rate of time of all the values suppressed.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state
INPUT_INVALID_ERROR	16#0C09	16#0001	"MeanFilterTControl" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0002	"SuppressionLimitHigh" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0003	"SuppressionLimitLow" value is lower than 0.0

Tab. 7-148: Generated error numbers of the IL_SuppressionType02 function block

7.8.8 IL_SuppressionType03

Brief description

This time-discreet FB suppresses measured values with an excessive deviation to the mean value of the measured values. The measured values outside the threshold values that can be parameterized are not transferred, but output as filtered value. The IL_SuppressionType03 is provided with functional extensions compared to Type01. Two of the extensions are the two switchable limits for the suppression window as well as an additional suppression function.

Interface description

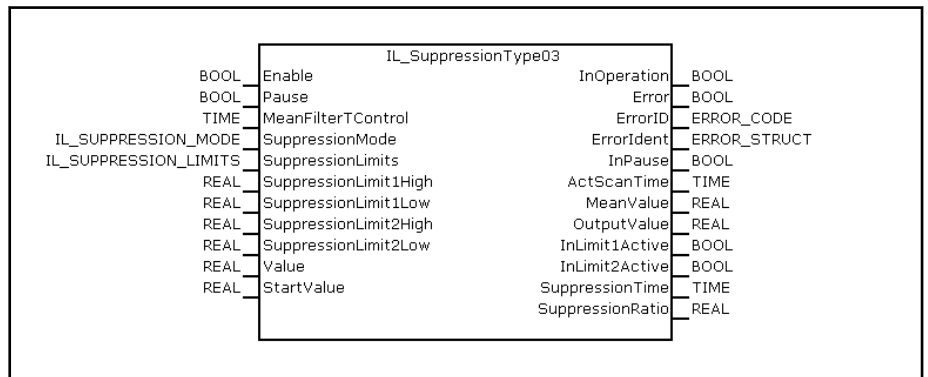


Fig. 7-92: IL_SuppressionType03 FB

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Calculation of data in each cycle while "Enable" = TRUE
	Pause	BOOL	The output data is not changed while "Pause" = TRUE
	MeanFilterTControl	TIME	Constant "Tn" of the (PT1-) mean value filter.
	SuppressionMode	IL_SUPPRESSION_MODE	Describes the function of the inputs "SuppressionLimitLow" and "SuppressionLimitHigh": - IL_SUPPRESSION_PERCENTAGE - IL_SUPPRESSION_ABSOLUTE_VALUE - IL_SUPPRESSION_STATISTICAL IL_SUPPRESSION_MODE, page 492
	SuppressionLimits	IL_SUPPRESSION_LIMITS	Selection of the active suppression window: - IL_SUPPRESSION_OFF - IL_SUPPRESSION_LIMIT_1 - IL_SUPPRESSION_LIMIT_2 IL_SUPPRESSION_LIMITS, page 491
	SuppressionLimit1High	REAL	Lower value of the suppression window #1.

	Name	Type	Comment
	SuppressionLimit1Low	REAL	Upper value of the suppression window #1.
	SuppressionLimit2High	REAL	Lower value of the suppression window #2.
	SuppressionLimit2Low	REAL	Upper value of the suppression window #2.
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
VAR_OUTPUT	InOperation	BOOL	Processing without errors, output data is valid.
	Error	BOOL	Error (see ErrorID and ErrorIdent)
	ErrorID	ERROR_CODE	Error description
	ErrorIdent	ERROR_STRUCT	For a detailed error description, see Error codes, page 452 .
	InPause	BOOL	Indicates that the "Pause" input is enabled and the output values do not change.
	ActScanTime	TIME	The time difference since the function block was called last time.
	MeanValue	REAL	Filtered mean value compared to the "Value".
	OutputValue	REAL	The calculated output value.
	InLimit1Active	BOOL	Indicates that the suppression window #1 is active.
	InLimit2Active	BOOL	Indicates that the suppression window #2 is active.
	SuppressionTime	TIME	Time counter, counts the time the measured values are suppressed. The value is deleted at the rising edge at "Enable" (T#0s) and counts up in case of an active suppression. As soon as an input value, which is not suppressed, is available again, the output is deleted again. In case of an overflow, the counter automatically becomes "0".
	SuppressionRatio	REAL	Percentage rate of the suppressed values (0-100%).

Tab. 7-149: IL_SuppressionType03 FB

IL_SUPPRESSION_MODE is an enumeration type and describes the function of the inputs "SuppressionLimitLow1/2" and "SuppressionLimitHigh1/2" with the following allocation:

Element	Description
IL_SUPPRESSION_PERCENTAGE	Values are on a percentage rate to the averaged input value
IL_SUPPRESSION_ABSOLUTE_VALUE	Values are absolute values to the averaged input value
IL_SUPPRESSION_STATISTICAL	Value are statistical to the default deviation σ of the input value (e.g. $0.5 \cdot \sigma$, $1.0 \cdot \sigma$, $4.0 \cdot \sigma$,...)

Tab. 7-150: Structure of the enumeration type IL_SUPPRESSION_MODE

IL_SUPPRESSION_LIMITS is an enumeration type and enables the suppression window with the following allocation:

Element	Description
IL_SUPPRESSION_OFF	Suppression is disabled and all values remain unchanged at the output.
IL_SUPPRESSION_LIMIT_1	Suppression window #1 active ("SuppressionLimit1Low" and "SuppressionLimit1High")
IL_SUPPRESSION_LIMIT_2	Suppression window #2 active ("SuppressionLimit2Low" and "SuppressionLimit2High")

Tab. 7-151: Structure of the enumeration type IL_SUPPRESSION_LIMITS

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
MeanFilterTControl	TIME	0.0	n. def	100 ms	Continuous
SuppressionMode	IL_SUPPRESSION_MODE			IL_SUPPRESSION_PERCENTAGE	Rising edge at Enable
SuppressionLimit	IL_SUPPRESSION_LIMITS			IL_SUPPRESSION_LIMIT_1	Continuous
SuppressionLimit1High	REAL	0.0	n. def	5.0	Continuous
SuppressionLimit1Low	REAL	0.0	n. def	5.0	Continuous
SuppressionLimit2High	REAL	0.0	n. def	20.0	Continuous
SuppressionLimit2Low	REAL	0.0	n. def	20.0	Continuous
Value	REAL	n. def	n. def	0.0	Continuous
StartValue	REAL	n. def	n. def	0.0	Rising edge at Enable

Tab. 7-152: Min./max. and default values of the IL_SuppressionType03 inputs

Functional description

When controlling, it is often required that the measured values with an obviously strong deviation from the expected value are filtered (suppressed). Therefore, a suppression window is implemented in the FB that moves along with the mean value of the value to be filtered and forms a tolerance window around this mean value. Each measured value "Value" is compared to the filtered measured value (filtering of the measured value with a PT1 filter). If the respective measured value is not within the suppression window formed around the mean value (tolerance window defined by "SuppressionLimit1/2Low" or "SuppressionLimit1/2High"), it is not output at the the FB output "OutputValue", but suppressed. That means that the mean value is output instead of "Value" at the "OutputValue" output.

The function is provided with two different windows between which can be switched. Additionally, the suppression function is deactivated. This can be used to increase the suppression window for a short period if it is operated with great deviations (e.g. in case of transient processes).

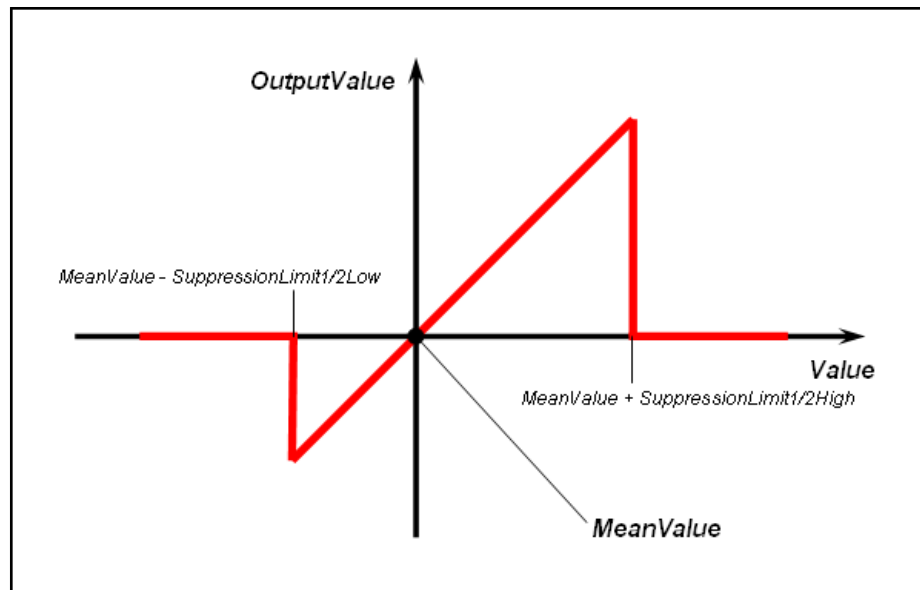


Fig. 7-93: Principle of the suppression window

Internal FB structure

The FB consists of an mean value filter and a block executing the suppression of the measured values. The mean value filtering is executed by a PT1 element. The filter time constant should be higher than the process time constant so that the mean value filtering can act correctly. The filtered mean value can be diagnosed via the "MeanValue" output.

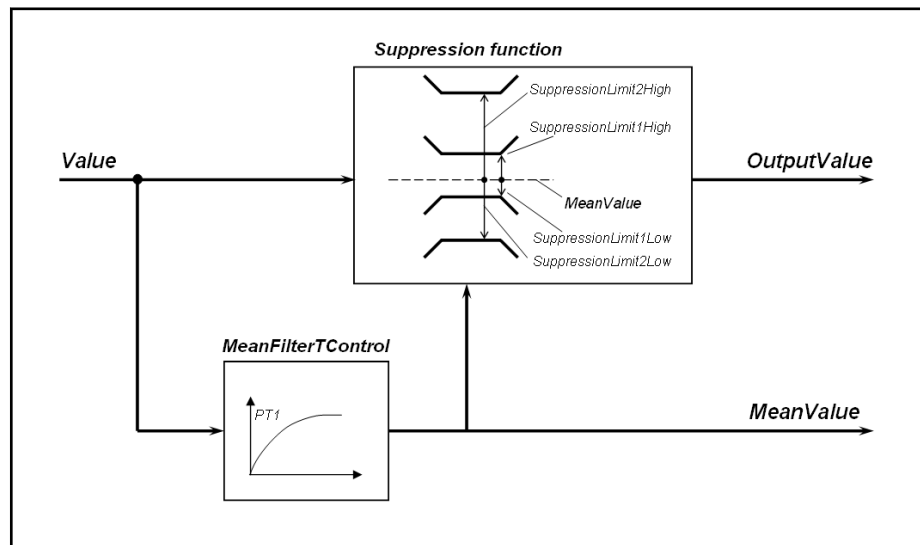


Fig. 7-94: Internal structure of the IL_SuppressionType03

Window-cutting mode

In case of window-cutting, the upper threshold ("SuppressionLimit1/2High") and the lower threshold ("SuppressionLimit1/2Low") can be configured separately. Three modes are available:

- **In percentage related to the mean value (IL_SUPPRESSION_PERCENTAGE):** the window width is specified by the percentage rate of the mean value. That means that "SuppressionLimitHigh" specifies a percentage rate (from the absolute value of the mean value) that is added to the mean value to maintain the upper threshold. "SuppressionLimitLow" specifies a percentage rate (from the absolute value of the mean

value) that is subtracted from the mean value for the lower threshold. It is to be considered that the window size might become very small if the amount of the mean value becomes very small.

- **Absolute window related to the mean value (IL_SUPPRESSION_ABSOLUTE_VALUE):** The values at the inputs "SuppressionLimitLow" and "SuppressionLimitHigh" directly specify the window size around the mean value.
- **Related to the statistical situation of the input signal (IL_SUPPRESSION_STATISTICAL):** the window width is related to the standard deviation σ of the measurement signal. That means that the standard deviation σ of the measurement signal with the values of the inputs "SuppressionLimit1/2Low" and "SuppressionLimit1/2High" is multiplied and results in the window size. (To determine the standard deviation, an exponentially floating calculation with "MeanFilterTControl" as time constant is used.)

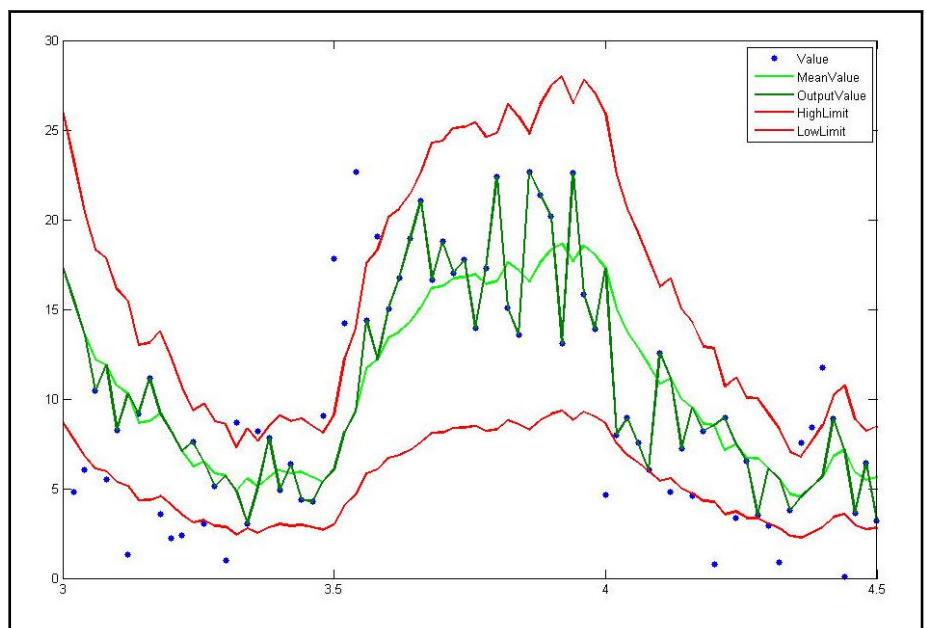


Fig. 7-95: Example of a window-cutting "related to the mean value in percentage"

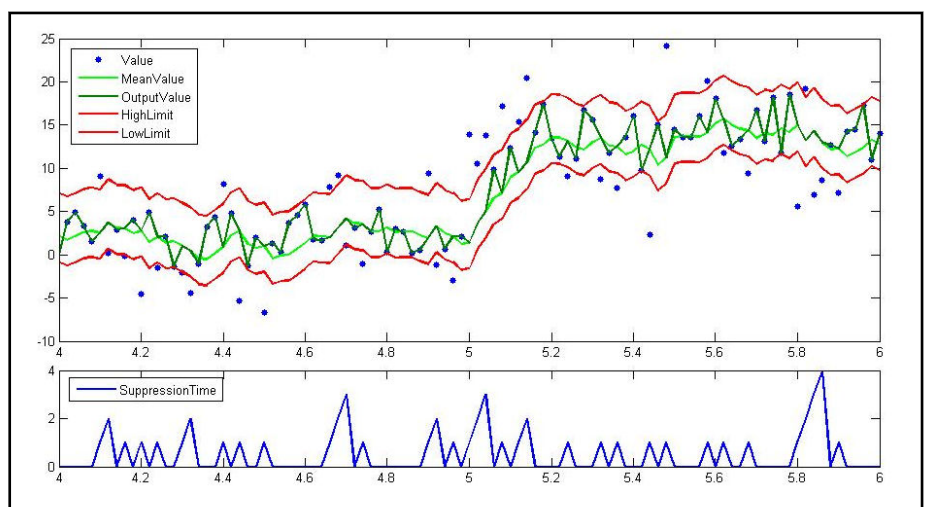


Fig. 7-96: Example of a window-cutting "with absolute window around the mean value"

Diagnostics of the window-cutting

The window-cutting function can be observed for diagnostic purposes via the outputs "SuppressionTime" and "SuppressionRatio". "SuppressionTime" is a time counter that counts the suppression period of the measured values. It counts up in case of an active suppression and is reset to "0" if an input value that is not suppressed is available. It is also reset to "0" in case of a rising edge at "Enable" as well as in case of an overflow. "SuppressionRatio" is the percentage rate of time of all the values suppressed.

Error codes The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state
INPUT_INVALID_ERROR	16#0C09	16#0002	"SuppressionLimit1/2High" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0003	"SuppressionLimit1/2Low" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0004	"MeanFilterTControl" value is lower than (2 x "ActScanTime"); only relevant in the "Statistical" mode

Tab. 7-153: Generated error numbers of the IL_SuppressionType03 function block

7.8.9 IL_SuppressionType04

Brief description This event-controlled FB suppresses measured values with an excessive deviation to the mean value of the measured values. The measured values outside the threshold values that can be parameterized are not transferred, but output as filtered value. The IL_SuppressionType04 is provided with functional extensions compared to Type02. Two of the extensions are the two switchable limits for the suppression window as well as an additional suppression function.

Interface description

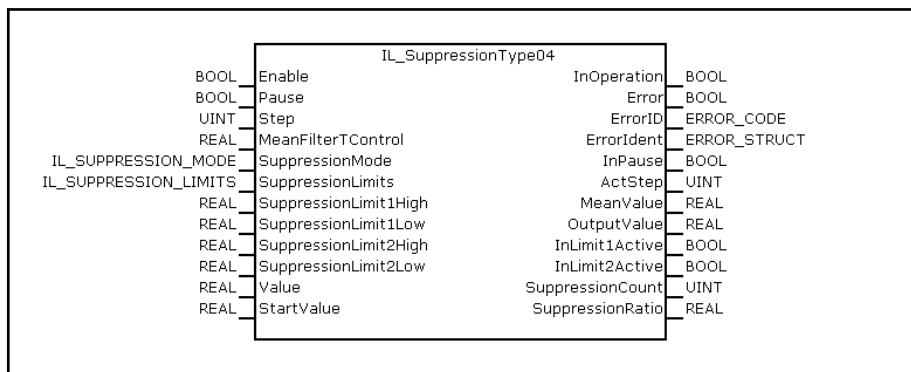


Fig. 7-97: IL_SuppressionType04 FB

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Calculation of data in each cycle (step) while "Enable" = TRUE
	Pause	BOOL	The output data is not changed while "Pause" = TRUE. If the "Step" input increases in the meantime, the step that is currently at the "ActSteps" output is issued.

	Name	Type	Comment
	Step	UINT	Current calculation step.
	MeanFilterTControl	REAL	The constant "Tn" of the mean value filter element related to "Step".
	SuppressionMode	IL_SUPPRESSION_MODE	Describes the function of the inputs "SuppressionLimitLow" and "SuppressionLimitHigh": - IL_SUPPRESSION_PERCENTAGE - IL_SUPPRESSION_ABSOLUTE_VALUE - IL_SUPPRESSION_STATISTICAL IL_SUPPRESSION_MODE, page 492
	SuppressionLimits	IL_SUPPRESSION_LIMITS	Selection of the active suppression window: - IL_SUPPRESSION_OFF - IL_SUPPRESSION_LIMIT_1 - IL_SUPPRESSION_LIMIT_2 IL_SUPPRESSION_LIMITS, page 491
	SuppressionLimit1High	REAL	Lower value of the suppression window #1.
	SuppressionLimit1Low	REAL	Upper value of the suppression window #1.
	SuppressionLimit2High	REAL	Lower value of the suppression window #2.
	SuppressionLimit2Low	REAL	Upper value of the suppression window #2.
	Value	REAL	The current actual value of the process.
	StartValue	REAL	The initial value, which is applied when at the rising edge at "Enable" at the "OutputValue" output.
VAR_OUTPUT	InOperation	BOOL	Processing without errors, output data is valid.
	Error	BOOL	Error (see ErrorID and ErrorIdent)
	ErrorID	ERROR_CODE	Error description
	ErrorIdent	ERROR_STRUCT	For a detailed error description, see Error codes, page 457 .
	InPause	BOOL	Indicates that the "Pause" input is enabled and the output values do not change.
	ActStep	UINT	Step value to which "OutputValue" refers. "ActStep" is initialized with the value of "Step" at a rising edge at "Enable".
	MeanValue	REAL	Filtered mean value compared to the "Value".
	OutputValue	REAL	The calculated output value.
	InLimit1Active	BOOL	Indicates that the suppression window #1 is active.
	InLimit2Active	BOOL	Indicates that the suppression window #2 is active.

	Name	Type	Comment
	SuppressionCount	UINT	Counter for the number of suppressed values. The value is deleted at the rising edge at "Enable" and counts up in case of an active suppression. As soon as an input value, which is not suppressed, is available again, the output is deleted again. In case of an overflow (16xFFFF), the counter automatically becomes "0".
	SuppressionRatio	REAL	Percentage rate of the suppressed values (0-100%).

Tab. 7-154: IL_SuppressionType04 FB

IL_SUPPRESSION_MODE is an enumeration type and describes the function of the inputs "SuppressionLimitLow1/2" and "SuppressionLimitHigh1/2" with the following allocation:

Element	Description
IL_SUPPRESSION_PERCENTAGE	Values are on a percentage rate to the averaged input value
IL_SUPPRESSION_ABSOLUTE_VALUE	Values are absolute values to the averaged input value
IL_SUPPRESSION_STATISTICAL	Value are statistical to the default deviation σ of the input value (e.g. $0.5 \cdot \sigma$, $1.0 \cdot \sigma$, $4.0 \cdot \sigma$,...)

Tab. 7-155: Structure of the enumeration type IL_SUPPRESSION_MODE

IL_SUPPRESSION_LIMITS is an enumeration type and enables the suppression window with the following allocation:

Element	Description
IL_SUPPRESSION_OFF	Suppression is disabled and all values remain unchanged at the output.
IL_SUPPRESSION_LIMIT_1	Suppression window #1 active ("SuppressionLimit1Low" and "SuppressionLimit1High")
IL_SUPPRESSION_LIMIT_2	Suppression window #2 active ("SuppressionLimit2Low" and "SuppressionLimit2High")

Tab. 7-156: Structure of the enumeration type IL_SUPPRESSION_LIMITS

Min./max. and default values of the inputs

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL			FALSE	Continuous
Pause	BOOL			FALSE	Continuous
Step	UINT	0	16#FFFF	0	Continuous
MeanFilterTControl	TIME	0.0	n. def	100 ms	Continuous
SuppressionMode	IL_SUPPRESSION_MODE			IL_SUPPRESSION_PERCENTAGE	Rising edge at Enable
SuppressionLimit	IL_SUPPRESSION_LIMITS			IL_SUPPRESSION_LIMIT_1	Continuous
SuppressionLimit1High	REAL	0.0	n. def	5.0	Continuous
SuppressionLimit1Low	REAL	0.0	n. def	5.0	Continuous
SuppressionLimit2High	REAL	0.0	n. def	20.0	Continuous
SuppressionLimit2Low	REAL	0.0	n. def	20.0	Continuous

Name	Type	Min. value	Max. value	Default value	Effective
Value	REAL	n. def	n. def	0.0	Continuous
StartValue	REAL	n. def	n. def	0.0	Rising edge at Enable

Tab. 7-157: Min./max. and default values of the IL_SuppressionType04 inputs

Functional description

When controlling, it is often required that the measured values with an obviously strong deviation from the expected value are filtered (suppressed). Therefore, a suppression window is implemented in the FB that moves along with the mean value of the value to be filtered and forms a tolerance window around this mean value. Each measured value "Value" is compared to the filtered measured value (filtering of the measured value with a PT1 filter). If the respective measured value is not within the suppression window formed around the mean value (tolerance window defined by "SuppressionLimit1/2Low" or "SuppressionLimit1/2High"), it is not output at the the FB output "OutputValue", but suppressed. That means that the mean value is output instead of "Value" at the "OutputValue" output.

The function is provided with two different windows between which can be switched. Additionally, the suppression function is deactivated. This can be used to increase the suppression window for a short period if it is operated with great deviations (e.g. in case of transient processes).

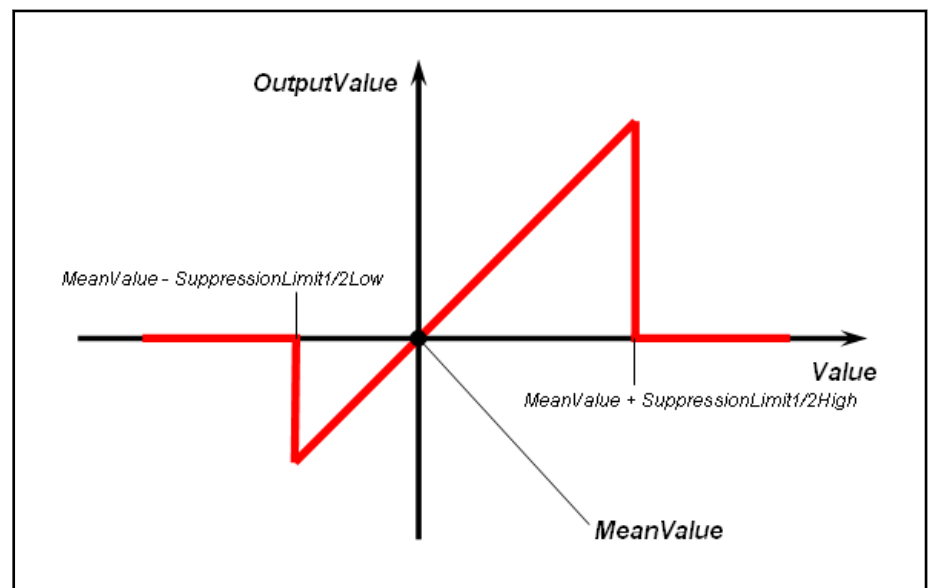


Fig. 7-98: Principle of the suppression window

Internal FB structure

The FB consists of an mean value filter and a block executing the suppression of the measured values. The mean value filtering is executed by a PT1 element. The filter time constant should be higher than the process time constant so that the mean value filtering can act correctly. The filtered mean value can be diagnosed via the "MeanValue" output.

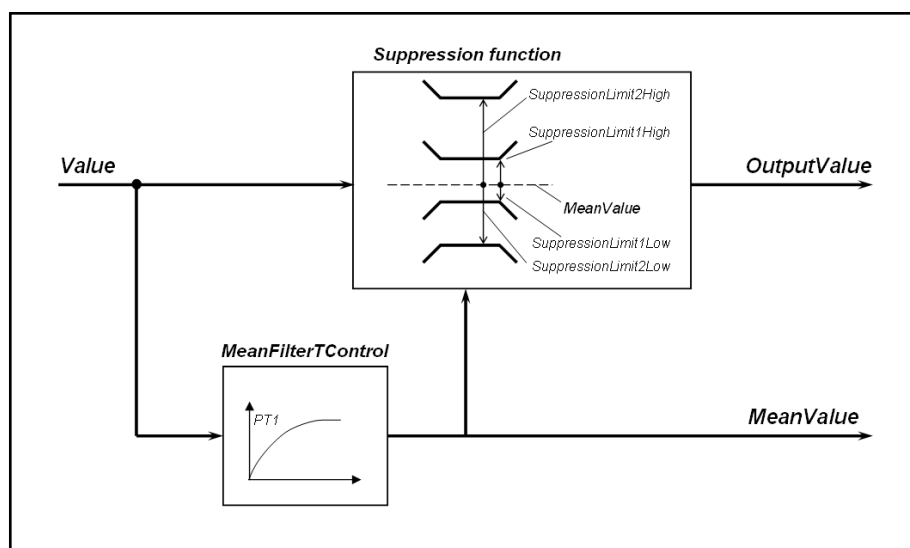


Fig. 7-99: Internal structure of the IL_SuppressionType02

Window-cutting mode

In case of window-cutting, the upper threshold ("SuppressionLimit1/2High") and the lower threshold ("SuppressionLimit1/2Low") can be configured separately. Three modes are available:

- **In percentage related to the mean value (IL_SUPPRESSION_PERCENTAGE):** the window width is specified by the percentage rate of the mean value. That means that "SuppressionLimitHigh" specifies a percentage rate (from the absolute value of the mean value) that is added to the mean value to maintain the upper threshold. "SuppressionLimitLow" specifies a percentage rate (from the absolute value of the mean value) that is subtracted from the mean value for the lower threshold. It is to be considered that the window size might become very small if the amount of the mean value becomes very small.
- **Absolute window related to the mean value (IL_SUPPRESSION_ABSOLUTE_VALUE):** The values at the inputs "SuppressionLimitLow" and "SuppressionLimitHigh" directly specify the window size around the mean value.
- **Related to the statistical situation of the input signal (IL_SUPPRESSION_STATISTICAL):** the window width is related to the standard deviation σ of the measurement signal. That means that the standard deviation σ of the measurement signal with the values of the inputs "SuppressionLimit1/2Low" and "SuppressionLimit1/2High" is multiplied and results in the window size. (To determine the standard deviation, an exponentially floating calculation with "MeanFilterTControl" as time constant is used.)

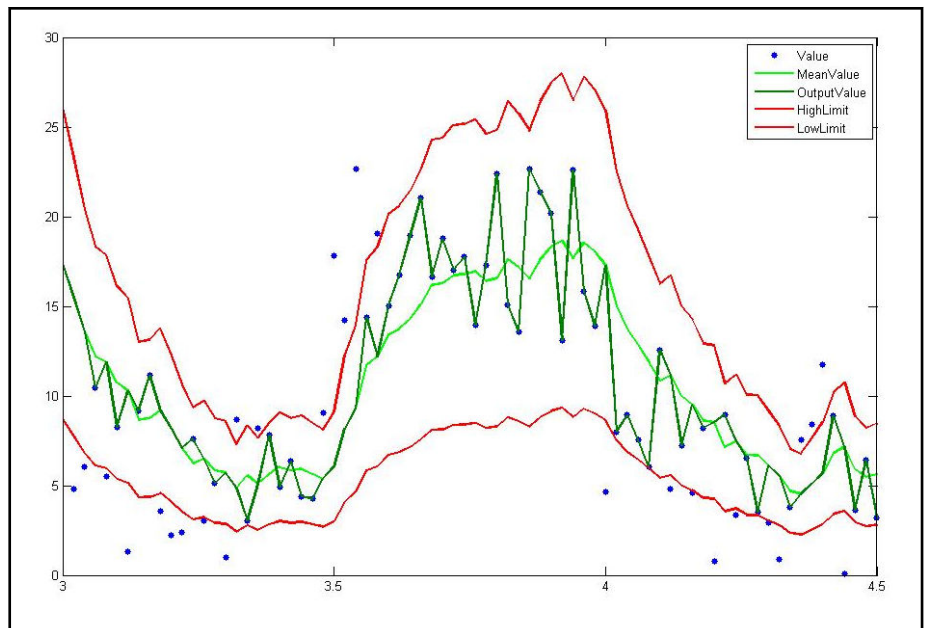


Fig. 7-100: Example of a window-cutting "related to the mean value in percentage"

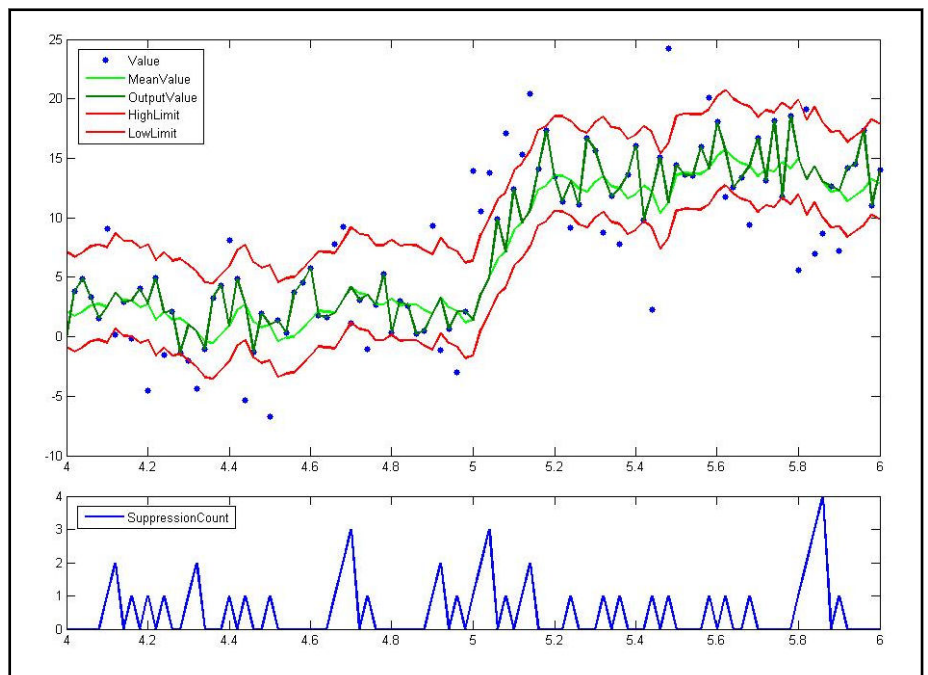


Fig. 7-101: Example of a window-cutting "with absolute window around the mean value"

Diagnostics of the window-cutting

The window-cutting function can be observed for diagnostic purposes via the outputs "SuppressionCount" and "SuppressionRatio". "SuppressionCount" is a counter that counts the suppression period of the measured values. It counts up in case of an active suppression and is reset to "0" if an input value that is not suppressed is available. It is also reset to "0" in case of a rising edge at "Enable" as well as in case of an overflow. "SuppressionRatio" is the percentage rate of time of all the values suppressed.

Error codes

The function block uses the error table **F_RELATED_TABLE**, 16#0170. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR (16#0005)	16#0006	16#0000	Invalid function block state
INPUT_INVALID_ERROR	16#0C09	16#0001	"MeanFilterTControl" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0002	"SuppressionLimit1/2High" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0003	"SuppressionLimit1/2Low" value is lower than 0.0
INPUT_INVALID_ERROR	16#0C09	16#0004	The "MeanFilterTControl" value has to be twice the difference of two (different) sequent "Step" values (only in the "Statistical" mode)

Tab. 7-158: Generated error numbers of the IL_SuppressionType04 function block

7.9 Temperature controller (TemperatureController)

7.9.1 Introduction and overview



The following function blocks and data types were in the version 13 in the ML_TechBase library.

They were moved to RIL_LoopControl library in version 14.

The following function blocks control temperature-controlled systems.

Areas of application and functions/function blocks of RIL_LoopControl

Area of application (folder in library)	
Function blocks	Description
Temperature controller (TemperatureController)	
IL_TempControlType01, page 458	Controller for the temperature-controlled system with automatic offline controller setting
IL_TempControlType02, page 476	Controller for the temperature-controlled system with automatic online controller setting

Tab. 7-159: Function blocks to control temperature-controlled systems

7.9.2 IL_TempControlType01

Brief description

The IL_TempControlType01 function block controls the temperature. It can determine the controller parameters either independently or with specifications. Special functions like monitoring, pause and manual mode can also be enabled.



In 14VRS, the "IL_TempControlType01 (ML_TechBase library)" was included in the "RIL_LoopControl library".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_LoopControl

Tab. 7-160: Reference table of the IL_TempControlType01 function block

Interface description

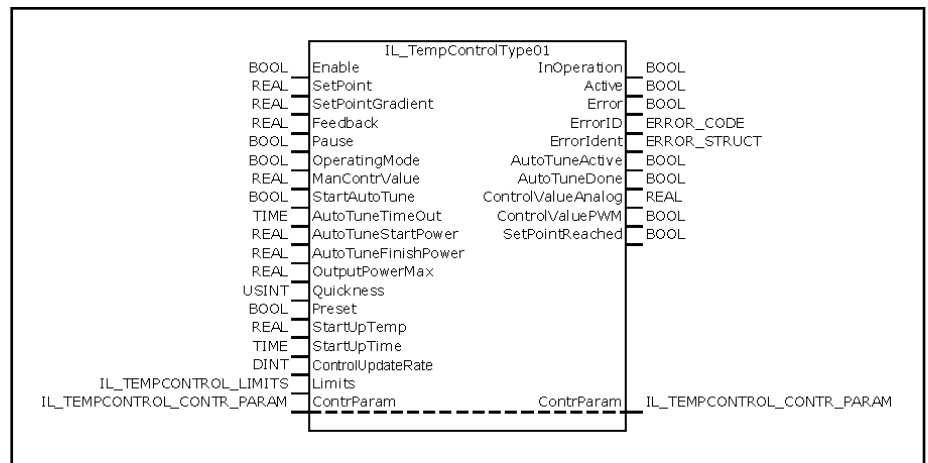


Fig. 7-102: IL_TempControlType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	The function block is enabled if this input is "TRUE". The function block is reset to its initial values with a falling edge. The input is read in each cycle
	SetPoint	REAL	Command temperature Unit in [u]* The input is read in each cycle
	SetPointGradient	REAL	Maximum parameter change "SetPoint" in [rps]. The input is read in each cycle. The preset value is "0 [rps]" (not active)
	Feedback	REAL	Input for the actual temperature Unit in [u] The input is read in each cycle
	Pause	BOOL	This input "freezes" the control variable and the controller calculation. The input is read in each cycle. The preset value is "FALSE" (not active)
	OperatingMode	BOOL	Sets the controller mode from automatic to manual mode: AUTO:=FALSE, MANUAL:=TRUE. The preset value is "FALSE" (AUTO). The input is read in each cycle
	ManContrValue	REAL	Specifies the control variable in [%] for the manual mode. Is only actively read if "OperatingMode:=1". The input is read in each cycle. The preset value is "0%" (no control variable)

I/O type	Name	Data type	Description
	StartAutoTune	BOOL	A rising edge starts the identification. The PID-controller parameters are calculated subsequently. They are not active until a falling edge is present. The input is read in each cycle. The preset value is "FALSE" (not active)
	AutoTuneTimeOut	TIME	Watchdog responsible for the time monitoring during the identification. The preset value is "$10000 \cdot T_A$". The input is read at a rising edge of "StartAutoTune".
	AutoTuneStartPower	REAL	Control variable in [%] representing the lower limit of the identification. The preset value is "0%". The input is read at a rising edge of "StartAutoTune".
	AutoTuneFinishPower	REAL	Control variable in [%] representing the upper limit of the identification. The preset value is "50%". The input is read at a rising edge of "StartAutoTune".
	OutputPowerMax	REAL	Maximum output control variable in [%]. The preset value is "100%". The input is read in each cycle
	Quickness	USINT	Quickness of the controller The preset value is "3" (very slight overshoot). The input can be selected in the range from "1" (quick control, strong overrun) to "5" (slow control, no overshoot). The input is read at a rising edge of "StartAutoTune".
	Preset	BOOL	A rising edge applies the controller parameters of the "ContrParam" structure. The preset value is "FALSE" (not active). The input is read during each cycle and becomes active only if "Enable" = TRUE
	StartUpTemp	REAL	Startup temperature in [u] The input is read at a rising edge of "StartAutoTune". The preset value is "0u" (not active). Becomes active if a value not equal to "0" is assigned to the "StartUpTime" input
	StartUpTime	TIME	Startup time, i.e. the period during which the startup temperature ("StartUpTemp") is not changed until the "normal" closed-loop control is started. The preset value is "T#0s" (not active). The input is read at a rising edge of "Enable"

I/O type	Name	Data type	Description
	ControlUpdateRate	DINT	This factor specifies how much slower the controller works compared to the task of the function block. The sampling time results from multiplying the PLC cycle time (= task interval) with "ControlUpdateRate"
	Limits	IL_TEMPCONTROL_LIMITS	Structure containing the alarm limits in [u] (units). IL_TEMPCONTROL_LIMITS, page 488
VAR_OUTPUT	InOperation	BOOL	The temperature control is active if an output is set
	Active	BOOL	This output is set if "Enable" is set and if the controller is active
	Error	BOOL	An error occurred at runtime. The control variable is set to "0%"
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, this output contains detailed error information Troubleshooting, page 473
	AutoTuneActive	BOOL	Identification is active
	AutoTuneDone	BOOL	Identification process completed successfully
	ControlValueAnalog	REAL	Analog value of the control variable in %
	ControlValuePWM	BOOL	Digital, pulse-width modulated control variable value; either "0" or "1"
	SetPointReached	BOOL	A status output indicates whether the actual value is within the limits ("Setpoint" + "Limits.UpperAtSetpoint") $\geq$ actual value $\geq$ (Setpoint - "Limits.LowerAtSetpoint")
VAR_IN_OUT	ContrParam	IL_TEMPCONTROL_CONTR_PARAM	This structure contains the current parameters K_P , K_I and K_D of the controller as well as the dead time T_t IL_TEMPCONTROL_CONTR_PARAM, page 486

- * The unit [u] means "unit". This can be any unit such as [°C] or [°F]. Ensure that the selected unit is always used.
- + The sampling time T_A is the effective sampling time of the temperature controller and consists of $T_A = \text{cycle time of the PLC task the function block is called in multiplied with "ControlUpdateRate"}$

Tab. 7-161: Interface variables of the IL_TempControlType01 function block

Time diagram



Fig. 7-103: Time diagram

Min./max. and default values

The following table lists the min./max. and default values as well as the application behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Cyclic
SetPoint	REAL	N. def.	N. def.	0.0	Cyclic
SetPointGradient	REAL	0.0	N. def.	0.0	Cyclic
Feedback	REAL	N. def.	N. def.	0.0	Cyclic
Pause	BOOL	FALSE	TRUE	FALSE	Cyclic
OperatingMode	BOOL	FALSE	TRUE	FALSE	Cyclic
ManContrValue	REAL	0.0	100.0	0.0	Cyclic
StartAutoTune	BOOL	FALSE	TRUE	FALSE	Cyclic
AutoTuneTime-Out	TIME	T#0s	"10000 • T _A ".	"10000 • T _A ". (is automatically set when T#0s is set)	At a rising edge at "StartAutoTune"
AutoTuneStart-Power	REAL	0.0	95.0	0.0	At a rising edge at "StartAutoTune"
AutoTuneFinish-Power	REAL	5.0	100.0	50.0	At a rising edge at "StartAutoTune"
OutputPowerMax	REAL	>0.0	100.0	100.0	Cyclic
Quickness	USINT	1	5	3	At a rising edge at "StartAutoTune"
Preset	BOOL	FALSE	TRUE	FALSE	Cyclic
StartUpTemp	REAL	N. def.	N. def.	0.0	At a rising edge at "Enable"
StartUpTime	TIME	T#0s	N. def.	T#0s	At a rising edge at "Enable"
ControlUpdateRate	DINT	1	1000	100	At a rising edge at "Enable"
Limits	IL_TEMPCONTROL_LIMITS	See "Data structure"	See "Data structure"	See "Data structure"	Cyclic
ContrParam	IL_TEMPCONTROL_CONTR_PARAM	See "Data structure"	See "Data structure"	See "Data structure"	Data is written by the autotune process at a rising edge at "AutoTuneDone". Data is applied at a rising edge at "Preset".

Tab. 7-162: Min./max. and default values as well as the transfer behavior of the inputs of the function block IL_TempControlType01

Application The function block can be used for theMLC system using the temperature modules (IL TEMP 2 RTD). Each value can be used as temperature value, since the temperature controller does not directly access the hardware. Thus, it is also possible to obtain temperature values via field bus connections for example.

Identification The automatic identification, also known as "AutoTune", identifies the parameters of the controlled system to determine the parameters of the controller.

First, certain model parameters are calculated. Subsequently, the model is compared to the real controlled system to avoid incorrect parameterizations of the controller parameters.

A command value jump is added to the control system. Before the jump is activated, it has to be ensured that the controlled system is in passive state. Therefore, a waiting time is started. The preset value for this time monitoring is 5,000 multiplied with the root of the sampling time " T_A " in [ms]. The actual value must not exceed the tolerance limits of the defined tolerances up to the end of the waiting time. Otherwise, the waiting time is restarted and the system waits until the specified timeout. The limit for the temperature window containing the actual value for the time of the waiting time is calculated as follows:

$$|\text{IdentTemp} - \text{ActTemp}| < \sqrt{\frac{|\text{IdentTemp}|}{10}} + 0.5$$

The variable "IdentTemp" is the value around which the tolerance window of the actual temperature is created. At the beginning, "IdentTemp" is initialized with the actual value ("IstTemp"). Waiting for the stationary final value after the jump is performed identically.

Example:

Estimation of the time required for identification

It is assumed, that the function block is operated in a PLC task with a cycle time of 10ms. The "ControlUpdateRate" input is connected to 100. What is the minimum time for the "Autotune" process?

The sampling time can be calculated multiplying the PLC task cycle time and the "ControlUpdateRate": " T_A " = PLC task cycle time * "ControlUpdateRate" = 10ms * 100 = 1000ms

Use the formula above to calculate the waiting time: Waiting time = 5000 * SQRT(" T_A ") = 158114ms or 2min and 38s. As it has to be waited for double the period for a jump response, a total waiting time of at least 5min and 16s results. The actual time required to reach the stationary state has to be added.

The quality of the determined model is estimated. If the deviation is too big, the function block outputs the error

"16#1603" (CALCULATION_ERROR)

with the identifier "16#0008" (invalid process model).

Setting the controller parameters

Quickness = 1...5;

1 = very fast control with large overshoot;

5 = very slow control without overshoot;

Preset value is quickness = 3.

The calculated controller parameters are entered into the provided structure "IL_TEMPCONTROL_CONTR_PARAM".

The entry

KP corresponds to the proportional gain factor

KI corresponds to the integration constant

KD corresponds to the differentiation constant and

Tt corresponds to the dead time

If these parameters are specified by the user, at least an estimated value is to be entered for the dead time to avoid the error

"Error in the control loop, sensor possibly displaced or destroyed"

(Additional1 = 16#1602; Additional2 = 16#0006),

Control algorithm

An additive PIDT₁ algorithm with anti-wind-up of the I-value and PT₁ filtering of the D-part is implemented. The additive form allows to disable certain parts of the controller. Thus, it can be switched to the structure of a PI-controller by parameterizing the D-value to "zero" for example.

The I-value is limited using an anti-wind-up delimiter which is internally preset to 90 % of the control variable.

The saturation limit of the controller is the maximum output value (VAR_IN OutputPowerMax) of the controller. The preset value is 100%.

Controller structure

In the following, the PIDT₁ controller structure with anti-wind-up limitation and limitation of the maximum control variable are displayed. Due to the additive structure, the values of the controller can be individually disabled. Thus, pure P-, P-I or PDT₁-controllers can be parameterized as well. However, the individual values have to be disabled by the user. For this purpose, the desired values have to be entered into the "contrParam" structure. At a rising edge at the "Preset" input, these new parameters are enabled in the controller. By default, the controller works with the complete PIDT₁ structure.

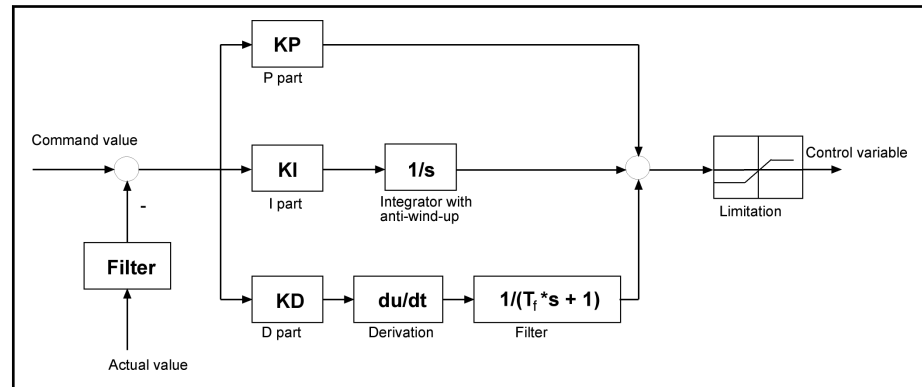


Fig. 7-104: Controller structure

The actual temperature value is smoothed with a PT₁ filter during the identification and during the control to avoid stochastic failures. The time constant of the filter is fixed to 20 times of the cycle time.

Control loop monitoring

The function block is provided with a control loop monitoring. If a control variable of "100%" is output (full power), the temperature has to change in the direction of the sign set in K_p. Otherwise, a timer is enabled displaying an error at timeout. The timer is reset as soon as the temperature moves into the correct direction. The timer is assigned with the following values:

If an I-value exists, (2000/K_I) + dead time is used. If an I-ratio is used, only the value 2000 * cycle time is preset. The timer is reset even if the control variable becomes again less than "100%". It is important to specify the dead time (T_t) in the structure for the controller parameters as well if the controller parameters are specified by the "Preset" input. If K_I is high and there is a high dead time, monitoring could be triggered otherwise. Due to this monitoring, sensor errors (e.g. by being shifted) or actuating element errors can be detected.

In addition, the function block also monitors the gradient of the actual value. However, this monitoring function becomes only active in case of large deviations. There can be deviations for example if the sensor displays an error code and if this error code is not caught/evaluated by the user. The gradient is set to 10 rms. A one-time exceedance is tolerated before an error is generated.

Using the function block

The function block should be operated in a cyclic task. The function block automatically detects the task interval (= cycle time) and adjusts the control algorithm accordingly. The controller works with the sampling time resulting from the task interval multiplied with the input variable "ControlUpdateRate". If "ControlUpdateRate" = 1, the sampling time corresponds to the task interval. The sampling time is used when identifying the controlled system and if the controller is enabled. All other functions of the function block (reading actual values, gradient monitoring, PWM output, manual closed-loop control) use the task interval. Thus, the control/identification process can be separated from other functions of the function block (with regard to time). This is especially important to separate the output of the control variable from the control/identification process using the PWM.

The pulse-width modulation (PWM) is based on the task interval, i. e. the task interval is the smallest possible time slice which can be set to "ON" or "OFF" during the PWM. One PWM period is 20 times longer than the task interval. Thus, the control variable can be changed in steps of 5 % between 0 % and 100 %.

Example:

The task interval is parameterized to 20 ms for the cyclic task. This means that a new value for the control variable can be transferred to the PWM every 400 ms. Thus, the possible switch-on/switch-off times for the PWM are:

Control variable [%]	Switch-on period [ms]	Switch-off period [ms]
0	0	400
5	20	380
10	40	360
...		
50	200	200
...		
95	380	20
100	400	0

Tab. 7-163: Example of switch on/off times

The task interval and "ControlUpdateRate" always have to be selected with regard to the control element and the controlled system. If the task intervals are too short, the control element wears very quickly (such as relay modules). If the task intervals/sampling times are too long, the accuracy of the closed-loop control reduces.

Furthermore, the sampling time generating and error if it exceeds/falls below by more than 20 % an setting the "Error" output to TRUE is monitored.

Recommendation for the task interval selection

Set the task interval/cycle time to ensure that the PWM does not cause any visible temperature changes in a PWM period of 20*task interval. If, the control variable is 20 % for example, the temperature of the controlled system

Recommendation for the sampling time

strongly increases during the heating and reduces while the heating is switched off. This means that the task interval is too long. In contrast, it is also not recommended to choose a task interval that is too short, since the control element is subject to higher loads (e. g. the relay module has to switch very often).

Tests determine an appropriate task interval.

First, select the task interval as described in the previous section. The sampling time can be set to the desired value using the "ControlUpdateRate" input.

The following sampling time is recommended:

- 10 ms for quick/dynamic controlled systems (time constant for the controlled system from 200 ms to 2 s)
- 100 ms for average controlled systems (time constant for the controlled system from 2 s to 20 s)
- 1 s to 30 s for slow/static controlled systems (time constant for the controlled system from 20 s to 100 min).

The sampling time should be approximately 20 to 200 times faster than the "dominating time constant" of the controlled system. The "dominating time constant" is approximately the time a controlled system requires to pass 63% of an added jump. However, the sampling time can be freely selected by the user.

If the selected sampling time is too short, the error "Time overflow during identification (watchdog)" (Additional1 = 16#1605; Additional2 = 16#0009) can occur during the identification. In this case, a longer sampling time has to be selected, since the controlled systems was not in steady state when the time monitoring elapsed.

If the selected sampling time is too long, the error "Identified controlled process parameters invalid, self-identification of the controlled process failed" (Additional1 = 16#1603; Additional2 = 16#0007) can occur during the identification. In this case, select a shorter sampling time, since the intervals the controlled system behavior was measured were too long.

Recommendation for the identification

For the identification, some rules are to be observed to achieve good results. These are explained in detail in the following.

In general, temperature-controlled system are not linear. For low temperatures, less power is required to achieve a certain temperature increase than at higher temperatures. At low temperatures, a controller needs different settings than at high temperatures.

Thus, it is particularly important to carry out the identification as well in the temperature range in which the controller is to operate later on. There are two options:

1. For a very precise closed-loop control in a small temperature range, the power required in the manual mode is to be specified first so that the desired temperature results. Subsequently, perform the identification in which the value for "AutoTuneStartPower" is a bit lower than the calculated value and the value for "AutoTuneFinishPower" is a bit higher than the calculated value. The controller parameters are determined precisely for this temperature range. However, outside the temperature range, the closed-loop control is less precise.
2. For a wide closed-loop control range, the power required for the lowest and highest temperature in manual mode is to be determined. These are used as values for "AutoTuneStartPower" and "AutoTuneFinishPower". The closed-loop control is averaged over the entire temperature

range. In general, this results in a more dynamic control at low temperatures and more diminished closed-loop control at higher temperatures.

Describing the individual functions

With regard to the following use cases, the variables have to be specified in the declaration part of the user program to assign the function block.

Program:

```

VAR
fb_TempControl:IL_TempControlType01;      (* Instance of the function blocks *)
strContrParam: IL_TEMPCONTROL_CONTR_PARAM; (* Structure containing the controller parameters *)
strLimits: IL_TEMPCONTROL_LIMITS;         (* Alarm/status limits *)
rActualTemp:REAL;                          (* Actual temperature*)
rActualPower:REAL;                        (* Control variable, analog *)
bActualPower:BOOL;                        (* Control variable, digital (PWM) *)
bAutoTuneDone:BOOL;                       (* Identification state *)
bSetPointReached:BOOL;                    (* Actual value is within the command value window *)
bError:BOOL;                              (* State of the error output *)
iState:INT;                               (* Value in SFC *)
END_VAR

```

For an easier representation, no errors are considered.

Simple closed-loop control with a controller parameter specification

In the "strContrParam" structure, the function block is parameterized with the desired controller parameters. At a rising edge at the "Preset" input, new controller parameters can also be parameterized during the closed-loop control. The command value ("Setpoint") has to be preset and the actual value ("Feedback") has to be returned. The control variable can either be retrieved at the analog output ("ControlValueAnalog") or at the digital output ("ControlValuePWM").

Program:

```

(* Setting the controller parameters *)
strContrParam.KP := 1.5;
strContrParam.KI := 2;
strContrParam.KD := 0.6;
strContrParam.Tt := T#1s;

(* Starting the function block at simultaneous transfer *)
(* of the controller parameters *)
fb_TempControl( Enable:=TRUE,
                Setpoint:=50,
                Feedback:=rActualTemp,
                Preset:=TRUE,
                ContrParam:=strContrParam,
                ControlValueAnalog=>rActualPower,
                ControlValuePWM=>bActualPower);

```

The controller parameters are only applied once and the controller works with the specified values.

Simple identification of a controlled system with a subsequent transition to the closed-loop control

First, the function block is set to the identification or AutoTune mode. After the successful identification, the closed-loop control is started by generating a falling edge at the input "StartAutoTune".

The calculated controller parameters are saved in the "strContrParam" structure and are thus available to the user. These controller parameters can be saved in the retain area of the control (refer to the example ["Retain backup of the controller parameters" on page 472](#)). Another identification is thus no longer required when starting the closed-loop control again. The simple control with parameter specification can be started immediately.

With the optional "Quickness" input, a slightly quicker setting was specified for the controller (quicker control but higher possibility for overshoots).

The identification is performed with a jump from 0 % ("AutoTuneStartPower") to 10 % ("AutoTuneFinishPower") of the output power

Program:

```

CASE iState OF
1: (* Identification *)
    fb_TempControl( Enable:=TRUE,
                    StartAutoTune := TRUE,
                    Feedback:=rActualTemp,
                    AutoTuneStartPower:= 0,
                    AutoTuneFinishPower:= 10,
                    Quickness:=2,
                    ContrParam:=strContrParam,
                    ControlValueAnalog=>rActualPower,
                    ControlValuePWM=>bActualPower,
                    AutoTuneDone=>bAutoTuneDone);
    IF (bAutoTuneDone = TRUE) THEN
        iState := 2;
    END_IF

2: (* Transition to closed-loop control *)
    fb_TempControl( Enable:=TRUE,
                    StartAutoTune := FALSE,
                    Setpoint:=50,
                    Feedback:=rActualTemp,
                    ContrParam:=strContrParam,
                    ControlValueAnalog=>rActualPower,
                    ControlValuePWM=>bActualPower);

```

Identifying the controlled system in the operating point

During a process control, another identification in the operating point can be desired. In the following, the corresponding procedure is described based on the previous example "Simple identification of a controlled process with subsequent transition to a closed-loop control". Setting the "AutoTuneStart" input starts the identification. The initial and final parameters of the control variable are transferred to "AutoTuneStartPower" and "AutoTuneFinishPower". After a successful identification, the new controller parameters are available in the "strContrParam" structure.

In the example, the operating range is calculated based on the actual value of the control variable (in %). The prerequisite is a controlled system in a steady (stationary) state. In the example, the operating range is selected from "control variable -10 %" to "control variable +10 %". The identification in the operating point starts when changing to state "3".

Program:

```

CASE iState OF
2: (* Closed-loop control *)
    fb_TempControl( Enable:=TRUE,
                    StartAutoTune := FALSE,
                    Setpoint:=50,
                    Feedback:=rActualTemp,
                    ContrParam:=strContrParam,
                    ControlValueAnalog=>rActualPower);
    If (bIdent = TRUE) THEN
        iState := 3;
    End_IF;

3: (* Identification in the operating point *)
    fb_TempControl( Enable:=TRUE,
                    StartAutoTune := TRUE,
                    AutoTuneStartPower:=rActualPower-10,
                    AutoTuneFinishPower:=rActualPower+10,
                    Feedback:=rActualTemp,
                    ContrParam:=strContrParam,
                    ControlValueAnalog=>rActualPower,
                    AutoTuneDone=>bAutoTuneDone);

    IF (bAutoTuneDone = TRUE) THEN
        iState := 2;
    END_IF

```

Defining a startup ramp

If a process prior to the actual temperature control requires a pause at a certain temperature for a certain period (startup ramp), this function is available via the inputs "StartUpTemp" and "StartUpTime".

First, the controller controls until the startup temperature. If this temperature is not achieved, the startup time "StartUpTime" elapses and the controller keeps the startup temperature constant. After the startup time "StartUpTime" has elapsed, the startup ramp is completed and the controller shows its normal behavior. The startup ramp is executed once after each change from "Enable" to "TRUE".

In the example, the controller controls to a temperature of "100". Then, it waits for 5 minutes and 30 seconds and controls it to the desired temperature of "150" (see [fig. 7-106 "Using a command value gradient and a startup ramp" on page 471](#)).

Program:

```
fb_TempControl(  Enable:=TRUE,
                  Setpoint:=150,
                  Feedback:=rActualTemp,
                  Preset:=TRUE,
                  StartUpTemp:=100,
                  StartUpTime:=t#5m30s,
                  ContrParam:=strContrParam,
                  ControlValueAnalog=>rActualPower);
```

Defining alarm and status limits

Alarm and status limits can be created for process monitoring purposes.

The "Limits" structure contains the values "UpperAlarm" and "LowerAlarm". When exceeding the alarm limits, an error is generated. The monitoring function is only enabled, when the current process temperature ranges between "UpperAlarm" and "LowerAlarm" in order not to generate an error during process start. These values are absolute values.

The structure "Limits" also contains the values "UpperAtSetPoint" and "LowerAtSetPoint". The "SetPointReached" is set as soon as the condition $(\text{Setpoint} - \text{Limits.LowerAtSetPoint}) \leq \text{Actual value} \leq (\text{Setpoint} + \text{Limits.UpperAtSetPoint})$ is met. These values are relative values that relate to the "Setpoint". The "Setpoint" can thus be modified without changing "Limits.LowerAtSetPoint" and "Limits.UpperAtSetPoint". Based on this status display, it can be determined whether the desired temperature has already been reached.

Program:

```
strLimits.UpperAlarm:=250;
strLimits.LowerAlarm:=70;
strLimits.UpperAtSetpoint:=20;
strLimits.LowerAtSetPoint:=20;

fb_TempControl(  Enable:=TRUE,
                  Setpoint:=150,
                  Feedback:=rActualTemp,
                  Limits:=strLimits,
                  ContrParam:=strContrParam,
                  ControlValueAnalog=>rActualPower,
                  SetPointReached=>bSetPointReached);
```

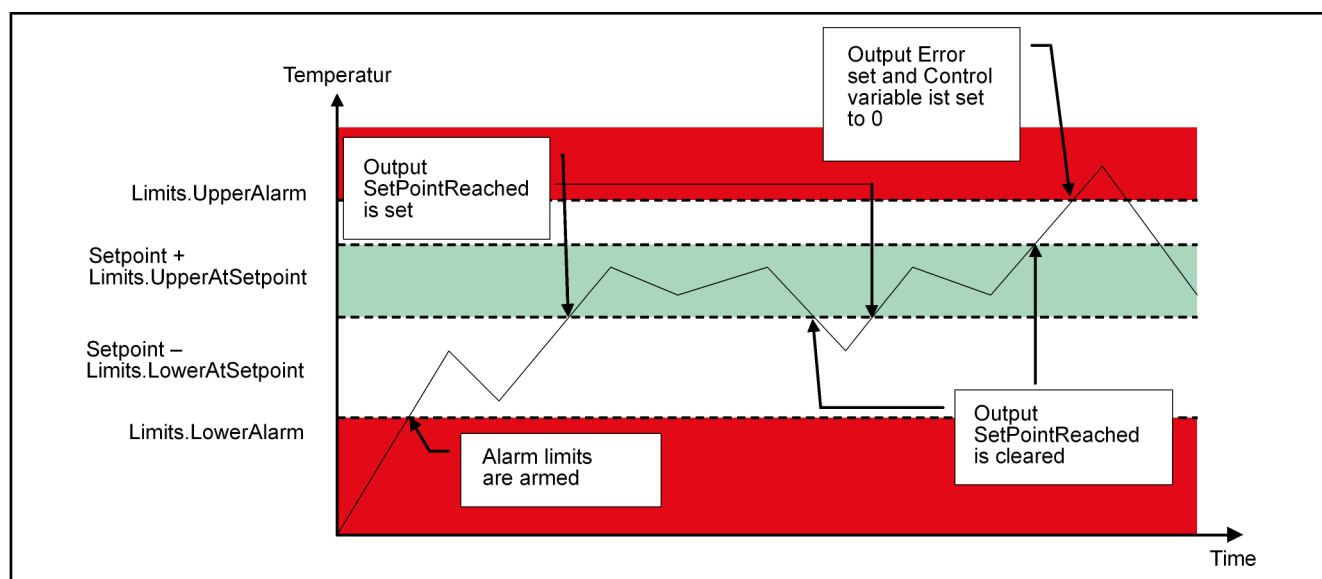


Fig. 7-105: Mode of operation of the alarm and status limits

Pausing the closed-loop control (PAUSE mode)

The closed-loop control can be paused via the "Pause" input. The control variable remains constant and the controller as well as all its values are "frozen". After the "Pause" input is reset, the closed-loop control continues.

In the example, the temperature is controlled to "150" in state "4". Switch to state 5 and the closed-loop control is "frozen" and no new control variables are output. Return to state "4" and the closed-loop control is continued and new control variables are output again. A jump can occur if the actual value has significantly changed in the meantime.

Program:

```

CASE iState OF
4:
fb_TempControl(  Enable:=TRUE,
                  Pause:=FALSE,
                  Setpoint:=150,
                  Feedback:=rActualTemp,
                  Preset:=TRUE,
                  ContrParam:=strContrParam,
                  Limits:=strLimits,
                  ControlValueAnalog=>rActualPower);

5:
fb_TempControl(  Enable:=TRUE,
                  Pause:=TRUE,
                  Setpoint:=150,
                  Feedback:=rActualTemp,
                  Preset:=TRUE,
                  ContrParam:=strContrParam,
                  Limits:=strLimits,
                  ControlValueAnalog=>rActualPower);

```

Analog/digital control variable output

The function block is provided with two control variable outputs: one analog output ("ControlValueAnalog") displaying REAL values between "0%" (no power) and "100%" (full power) and one digital output ("ControlValuePWM") displaying the BOOL values "FALSE" (no power) or "TRUE" (full power). The digital output can pulse the power when using relay modules. Pulsing takes place using a pulse-width modulation whose cycle time is 20 times slower than the time of the cyclic task in which the function block is running. Thus, control variables can be modulated in steps of 5 % with a resolution between 0 % to 100 %.

The analog output can be used for an analog output or to monitor the current control variable.

Program:

```
fb_TempControl(  Enable:=TRUE,
                 Pause:=FALSE,
                 Setpoint:=150,
                 Feedback:=rActualTemp,
                 Preset:=TRUE,
                 ContrParam:=strContrParam,
                 Limits:=strLimits,
                 ControlValueAnalog=>rActualPower
                 ControlValuePWM=>bActualPower);
```

Specifying a command value gradient

fb_TempControl(Enable:=TRUE, Pause:=FALSE, Setpoint:=150, Feedback:=rActualTemp, Preset:=TRUE, ContrParam:=strContrParam, Limits:=strLimits, ControlValueAnalog=>rActualPower ControlValuePWM=>bActualPower);

The function block can limit (ramp) the increase of the command value. This reduces the overshoot width and prevents the unit from damages. For each new specification of the input "SetPoint", the command value is internally changed in ramps (not jumps) until the desired command value is reached. "SetPointGradient" is entered in [rps].

In the example, the command temperature is controlled to "150" with an increase of "0.5 per second".

Program:

```
fb_TempControl(  Enable:=TRUE,
                 Pause:=FALSE,
                 Setpoint:=150,
                 Setpointgradient:=0.5,
                 Feedback:=rActualTemp,
                 Preset:=TRUE,
                 ContrParam:=strContrParam,
                 Limits:=strLimits,
                 ControlValueAnalog=>rActualPower);
```

Ramping with command value gradient and an additional startup ramp shows the following development of the command and actual value. It has been controlled to a command temperature ("Setpoint:=200") of "200°C" with a gradient ("Setpointgradient:=1") of "1°C/s". The startup temperature ("StartUpTemp:=100") is "100°C" and the startup time ("StartUpTime:=T#50s") is "50s".

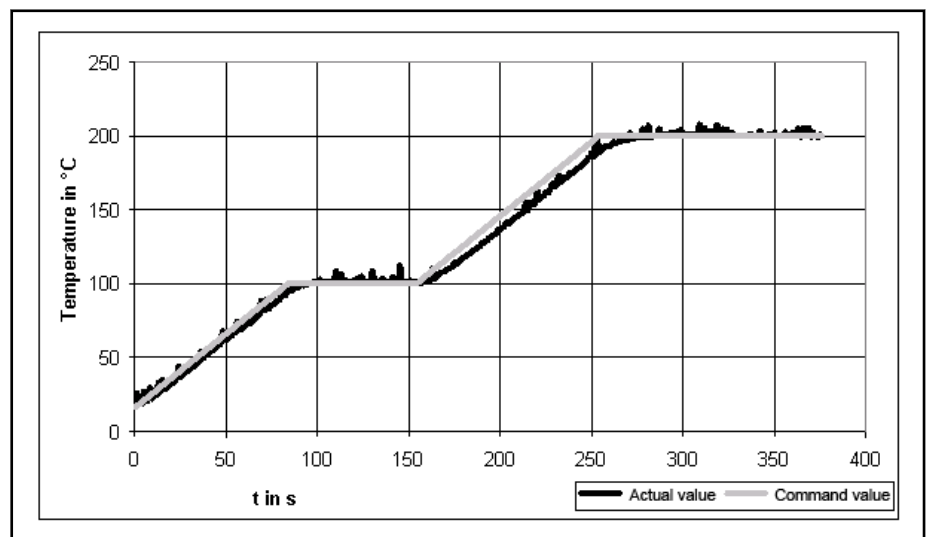


Fig. 7-106: Using a command value gradient and a startup ramp

Specifying a manual control variable

A manual control variable can be defined to control the temperature manually. At the same time, this variable represents the control variable at the func-

tion block output. A manual control variable can be required to record manual testing functions or to enable an emergency mode in case of a sensor error.

In the example, the function block is manually controlled with a control variable of "50%".

Program:

```
b_TempControl (  Enable:=TRUE,
                  Pause:=FALSE,
                  OperatingMode:=TRUE,
                  ManContrValue:=50,
                  ContrParam:=strContrParam,
                  Limits:=strLimits,
                  ControlValueAnalog=>rActualPower);
```

Retain backup of the controller parameters

To ensure that the controller parameters are safely stored even in the case of a power failure, they can be backed up in the retain area. Thus, no identification is required when switching on the controller and the closed-loop control can be started immediately.

For this purpose, create the "strContrParam" structure as retain variables ("VAR RETAIN") containing the values of the individual controller parameters. At the same time, create a mechanism to differentiate at the initial startup whether an identification has already been performed.

Additional retain variables have to be created in the header.

Program:

```
VAR RETAIN
    strContrParam: STRUCT_CONTR_PARAM;
    bIdentification: BOOL := FALSE;
END_VAR

If (bIdentification = FALSE) THEN
    fb_TempControl(  Enable:=TRUE,
                    StartAutoTune := TRUE,
                    Feedback:=rActualTemp,
                    Quickness:=2,
                    ContrParam:=strContrParam,
                    ControlValueAnalog=>rActualPower,
                    ControlValuePWM=>bActualPower,
                    AutoTuneDone=>bAutoTuneDone);

    IF (bAutoTuneDone = TRUE) THEN
        bIdentification:= TRUE;
    END_IF
ELSE
    fb_TempControl(  Enable:=TRUE,
                    StartAutoTune := FALSE
                    Preset:=TRUE,
                    Setpoint:=150,
                    Feedback:=rActualTemp,
                    ContrParam:=strContrParam,
                    Limits:=strLimits,
                    ControlValueAnalog=>rActualPower);

END_IF;
```

In the example above, the values of the controller are stored in the retain variables in each cycle. After a deactivation/power failure, the code checks if there has already been an identification using the Boolean variable "bIdentification". If this is not the case, an identification is performed. Otherwise, the controller can be restarted with the backed up (old) parameters. A rising edge has to be present at the input "Preset".

Troubleshooting with the substitute value strategy

By default, the function block displays the control variable "0%" (no power) in the event of an error. This is done for the system safety. If the user wants to use a control with a certain power, for example to avoid a strong cool down of a system, it is recommended to use the substitute value strategy described in the following example.

Errors can be checked in state "6" with "bError". If this condition is "TRUE", the function block is reset ("Enable:=FALSE") and state "7" is enabled.

In state "7", the function block is restarted with a rising edge at the input "Enable" and operated in manual mode (see ["Specifying a manual control variable" on page 471](#)).

In the example, a control variable of "50%" is output.

Further information on the error is output at the outputs "ErrorID" and "ErrorID-ident" which were not analyzed in this example.

Program:

```

CASE iState OF:
6:
fb_TempControl(  Enable:=TRUE,
                  Setpoint:=50,
                  Feedback:=rActualTemp,
                  Preset:=TRUE,
                  ContrParam:=strContrParam,
                  Limits:=strLimits,
                  ControlValueAnalog=>rActualPower,
                  Error=>bError);

IF (bError = TRUE) THEN
  fb_TempControl( Enable:=FALSE,
                  ContrParam:=strContrParam,
                  Limits:=strLimits);

  iState := 7;
END_IF

7:
fb_TempControl(  Enable:=TRUE,
                  Pause:=FALSE,
                  OperatingMode:=TRUE,
                  ManContrValue:=50,
                  ContrParam:=strContrParam,
                  Limits:=strLimits,
                  ControlValueAnalog=>rActualPower);

```

It has to be stated that the control variable output in the example above is set to 0 for one cycle (if the error is detected and if Enable:=FALSE). In state "7", the manual value is used as control variable.

Troubleshooting

The function block generates the following error messages in Additional1/Additional2 for the table **F_RELATED_TABLE**, 16#0170:

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#1600	Current state	Error in the state machine; invalid state
INPUT_RANGE_ERROR	16#1601	16#0001	Input values for a maximum output value ("OutputPowerMax", "AutoTuneStartPower" or "AutoTuneFinishPower") not in the valid range. Value has to be higher than or equal to 0 [%] up to max. 100 [%]
		16#1602	Input values for alarm limits invalid (Condition: Limits.UpperAlarm > Limits.LowerAlarm)
		16#1603	Input value for manual control variable invalid. Value has to be between 0[%] and 100[%]
		16#0004	Controller parameter invalid
		16#000A	Initial and final value for identification purposes are too close to each other. The minimum difference is 5[%]
		16#000B	The "SetpointGradient" input is invalid. Value has to be greater than 0
		16#000C	Input values invalid for status limits (Condition: (Setpoint + Limits.UpperAtSetPoint) > (Setpoint - Limits.LowerAtSetPoint))
		16#000D	Time for identification is too long ("AutoTuneTimeOut")
		16#000E	Value for "Quickness" is not in the valid limits (1 - 5)
		16#000F	Value for "ControlUpdateRate" is not in the valid limits (1 - 1000)
SYSTEM_ERROR	16#1602	Sampling time	Jittering the sampling time > 20[%]. Error becomes only active after two successive cycle timeouts
		16#0005	Actual value gradient is too high; possibly due to sensor error (diagnostics)
		16#0006	Error in the control loop, sensor possibly displaced or destroyed
CALCULATION_ERROR	16#1603	16#0007	Identified controlled system parameters are invalid, self-identification of the controlled system failed
		16#0008	The identified process model is not correct/invalid
ACCESS_ERROR	16#1604	Field number of the array	Incorrect access to internal variable

ErrorID	Additional1	Additional2	Description
OTHER_ERROR	16#1605	16#0009	Time overflow during identification (watchdog)
DEVICE_ERROR	16#1606	Temperature	Temperature error limit exceeded; temperature not in the error limits

Tab. 7-164: Error codes of the IL_TempControlType01 function block

Information on specific errors

- Error "Gradient of the actual value is too high"**
 (Additional1 = 16#1602; Additional2 = 16#0005):
 The gradient of the actual value is monitored. If the difference between two consecutive actual values is higher than 10 rms, this error is generated. This can be the case when using an Inline temperature model, since error analyses are displayed as measured values with values higher than 32768 (bit 15 set). If this diagnostics is not evaluated by the user, this error message can be output.
- Error "Error in the control loop, sensor possibly displaced or destroyed"**
 (Additional1 = 16#1602; Additional2 = 16#0006):
 It is monitored whether the actual value approaches the command value at a maximum output performance. For this purpose, time monitoring is started which is at an active I-value
 $2000/K_i + T_t$ and without the I-value is
 $2000 \cdot \text{sampling time} + T_t$.
 If the actual value does not approach the command value during this period, the error is generated. The sensor might be defective/displaced or the heating/cooling system does not work. The error can also occur if no dead time (T_t) is specified when specifying the controller parameters via the structure "IL_TEMPCONTROL_CONTR_PARAM". In this case, the value of the dead time is to be estimated and the controller parameters are to be set again. This is performed automatically when identifying the controlled system.
- Error "Identified controlled system parameters are invalid, self-identification of controlled system failed"**
 (Additional1 = 16#1603; Additional2 = 16#0007):
 The controlled system parameters resulted in invalid values. This can result due to an incorrect sampling time or incorrect actual values (sensor not connected). Also refer to ["Recommendation for the sampling time" on page 466](#).
- Error "The identified process model is not correct/invalid"**
 (Additional1 = 16#1603; Additional2 = 16#0008):
 When identifying the controlled system, the calculated model is compared to the measured values. The comparison did not show any satisfying results. The model shows a deviation of more than 20 % from the measured values. The sampling time chosen might have been incorrect or the controlled system does not show an aperiodic behavior. This can also be caused by disturbance variables occurring during the identification.
- Error "Time elapsed during identification (watchdog)"**
 (Additional1 = 16#1605; Additional2 = 16#0009):
 When identifying the controlled system, the system waits until the actual value is in a steady state. Subsequently, a command value step change

is performed and the system waits again until the actual value is in the steady state. If this requires too much time, the error is generated. The reason is either a strongly varying actual value or an incorrectly selected sampling time (see also next section)

- Error "Time for identification is too long ('AutoTuneTimeOut')"
(Additional1 = 16#1601; Additional2 = 16#000D):
When identifying the controlled system, the time monitoring can be set. This value can have a maximum of 10,000 * sampling time. In this case, the time monitoring has to be set to 0 (maximum possible time for identification at a given sampling time) or the sampling time has to be increased accordingly

7.9.3 IL_TempControlType02

Brief description The IL_TempControlType02 function block controls temperatures such as the [IL_TempControlType01, page 458](#) function block. It can determine the controller parameters either independently or using specifications.

The advantages of the two variants are:

- Shorter identification period (StartAutoTune)
- Changed controller parameterization (preset)
- Changed fine controller adjustment (DampingFactor)
- Input/output of the controlled system parameters (ModelParam)
- Individual controller design (ControlMode)

 In 14VRS, the "IL_TempControlType02 (ML_TechBase library)" was included in the "RIL_LoopControl library".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_LoopControl

Tab. 7-165: Reference table of the IL_TempControlType02 function block

Interface description

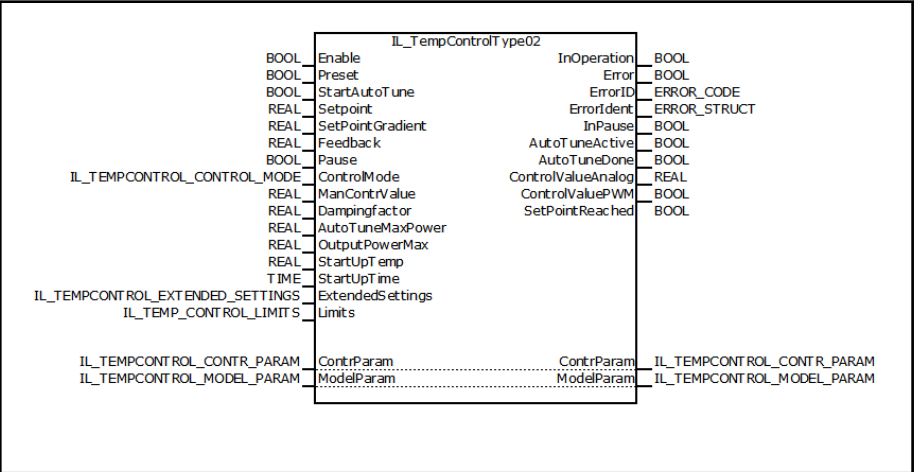


Fig. 7-107: IL_TempControlType02 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	The function block is enabled if this input is "TRUE". The function block outputs are reset to their default value with a falling edge.
	Preset	BOOL	A rising edge at this input applies the controller or controlled system parameters of the structures "ContrParam" and/or "ModelParam"
	StartAutoTune	BOOL	A rising edge at this input starts the self-adjustment. If this input is set together with "Enable", the controller parameters do not have to be specified
	SetPoint	REAL	Command temperature. Unit [u]*
	SetPointGradient	REAL	Maximum change of the "Setpoint" parameter. Unit [u/s]
	Feedback	REAL	Actual temperature. Unit [u]
	Pause	BOOL	This input "freezes" the control variable and the controller calculation
	ControlMode	IL_TEMPCONTROL_CONTROL_MODE	Selection of the control loop response. Either the open-loop control in the open control loop or the closed-loop control optimize the command value response or the disturbance response in the closed control loop IL_TEMPCONTROL_CONTROL_MODE, page 486
	ManContrValue	REAL	This input specifies the control variable when controlling manually. Unit [%]
	DampingFactor	REAL	This input affects the control loop response. It can be steplessly selected between a slow aperiodic and a fast oscillating response. Unit [-]
	AutoTuneMaxPower	REAL	Control variable limitation at the beginning of the self-adjustment. Unit [%]
	OutputPowerMax	REAL	Limitation of the controller output. If a control signal limitation in the controlled system is smaller than the limitation of the controller output, the anti-windup function of the controller can be restricted. Unit [%]
	StartUpTemp	REAL	Startup temperature. Is only active if a value not equal to 0 is assigned to the "StartUpTime" input. Unit [u]
	StartUpTime	TIME	Startup time, i.e. the period during which the startup temperature ("StartUpTemp") is not changed until the "normal" closed-loop control is started. The preset value is T#0s (not active)

I/O type	Name	Data type	Description
	ExtendedSettings	IL_TEMPCONTROL_EXTENDED_SETTINGS	Structure containing the extended setting options of the self-adjustment. The default settings are sufficient for the common controlled systems IL_TEMPCONTROL_EXTENDED_SETTINGS, page 487
	Limits	IL_TEMPCONTROL_LIMITS	Structure containing the alarm limits in [u] (units). IL_TEMPCONTROL_LIMITS, page 488 Unit [u]
VAR_OUTPUT	InOperation	BOOL	This output is set if "Enable" is set and if the function block is active
	Error	BOOL	An error occurred at runtime. The control variable is set to 0 [%]
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification
	ErrorIdent	ERROR_STRUCT	If the "Error" output is set, the output contains detailed error information Troubleshooting, page 484
	InPause	BOOL	The "ControlValueAnalog" output is "frozen" and the "ControlValuePWM" output remains active with this value
	AutoTuneActive	BOOL	The self-adjustment is active
	AutoTuneDone	BOOL	The self-adjustment was successfully completed
	ControlValueAnalog	REAL	Analog value of the control variable. Unit [%]
	ControlValuePWM	BOOL	Digital, pulse width-modulated control variable value. Unit [-]
	SetPointReached	BOOL	Status output whether the actual value is within the limits (Setpoint " + Limits.UpperAtSetpoint) >= Actual value >= (Setpoint - Limits.LowerAtSetpoint)
VAR_IN_OUT	ContrParam	IL_TEMPCONTROL_CONTR_PARAM	This structure contains the current parameters K_P , K_I and K_D of the controller as well as the dead time T_t IL_TEMPCONTROL_CONTR_PARAM, page 486
	ModelParam	IL_TEMPCONTROL_MODEL_PARAM	This structure contains the parameters K_S and T_S of the controlled system IL_TEMPCONTROL_MODEL_PARAM, page 489

*

The unit [u] means "unit". This can be any unit such as [°C] or [°F]. Ensure that the selected unit is always used.

Tab. 7-166: Interface variables of the IL_TempControlType02 function block

Min./max. and default values

The following table lists the min./max. and default values as well as the application behavior of the function block inputs:

Name	Type	Min. value	Max. value	Default value	Effective
Enable	BOOL	FALSE	TRUE	FALSE	Cyclic
Preset	BOOL	FALSE	TRUE	FALSE	Cyclic
StartAutoTune	BOOL	FALSE	TRUE	FALSE	Cyclic
SetPoint	REAL	N. def.	N. def.	0	Cyclic
SetPointGradient	REAL	0	N. def.	0	Cyclic
Feedback	REAL	N. def.	N. def.	0	Cyclic
Pause	BOOL	FALSE	TRUE	FALSE	Cyclic
ControlMode	IL_TEMPCONTROL_CON- TROL_MODE	CON- TROL_TEMP- MODE_MANUAL	CON- TROL_TEMP- MODE_ON- LY_DISTURB- ANCE	CON- TROL_TEMP- MODE_ON- LY_DISTURB- ANCE	Cyclic
ManContrValue	REAL	0	100	0	Cyclic
DampingFactor	REAL	20	200	100	Cyclic
AutoTuneMax- Power	REAL	>0	100	80	At a rising edge at "StartAutoTune"
OutputPowerMax	REAL	>0	100	100	Cyclic
StartUpTemp	REAL	N. def.	N. def.	0	At a rising edge at "Enable"
StartUpTime	TIME	T#0s	N. def.	T#0s	At a rising edge at "Enable"
ExtendedSettings	IL_TEMPCONTROL_EXTEN- DED_SETTINGS	See "Data struc- ture"	See "Data struc- ture"	See "Data struc- ture"	See "Data structure"
Limits	IL_TEMPCONTROL_LIMITS	See "Data struc- ture"	See "Data struc- ture"	See "Data struc- ture"	See "Data structure"
ContrParam	IL_TEMPCONTROL_CON- TR_PARAM	See "Data struc- ture"	See "Data struc- ture"	See "Data struc- ture"	See "Data structure"
ModelParam	IL_TEMPCONTROL_MOD- EL_PARAM	See "Data struc- ture"	See "Data struc- ture"	See "Data struc- ture"	See "Data structure"

Tab. 7-167: Min./max. and default values as well as the application behavior of the inputs of the function block IL_TempControlType02

Functional description

The function block can be operated at the MLC system. The actual temperature to be controlled can be determined using Inline temperature modules (IL TEMP 2 RTD) or provided differently via a field bus connection for example).

Except for the mode of operation of the auto tuning "StartAutoTune", the controller parameterization "Preset", the controller fine adjustment "DampingFactor", the input/output structure "ModelParam" and the option of the controller design "ControlMode", the IL_TempControlType02 function block is identical to the function block [IL_TempControlType01, page 458](#).

The internal PID-controller can either be parameterized as P-, PI-, PD- or PID-controller or the function block can automatically determine "StartAutoTune" controller parameters via an auto-tuning.

Controller/controlled system parameters

Compared to the IL_TempControlType01 function block, the identification of the self-adjusting temperature controller function block IL_TempControlType02 was modified to ensure that the control loop is now closed when starting the self-adjustment and that the controller parameters follow the currently identified controlled system parameters continuously.

For this functionality, set the "StartAutoTune" input with the controller design procedure "ControlMode".

The self-adjustment is not required for known controller or controlled system parameters. In this case, the values are specified via the inputs "ContrParam" and/or "ModelParam" and applied with a rising edge at the "Preset" input.

A PT_1 -filter continuously smoothes the actual temperature value ("Feedback") in the function block to reduce disturbances. The time constant of the filter is nearly 20 times higher than the cycle time ($T_{\text{Filt}}=20 \cdot T_{\text{Task}}$).

Due to this and further secondary delays caused by the process, the temperature controlled system can be assumed as non-oscillating PT_2 controlled systems with a dominant compensation constant T_s and a secondary delay T_t .

$$\frac{T_{\text{TC}}(s)}{u_{\text{TC}}(s)} \approx \frac{K_s}{(1+s \cdot T_s) \cdot (1+s \cdot T_t)}$$

The following jump response of a temperature-controlled system is an example assuming that the controlled system parameter shows $K_s = 1^\circ\text{C}/\%$, $T_s=10$ s and $T_t = 1$ s.

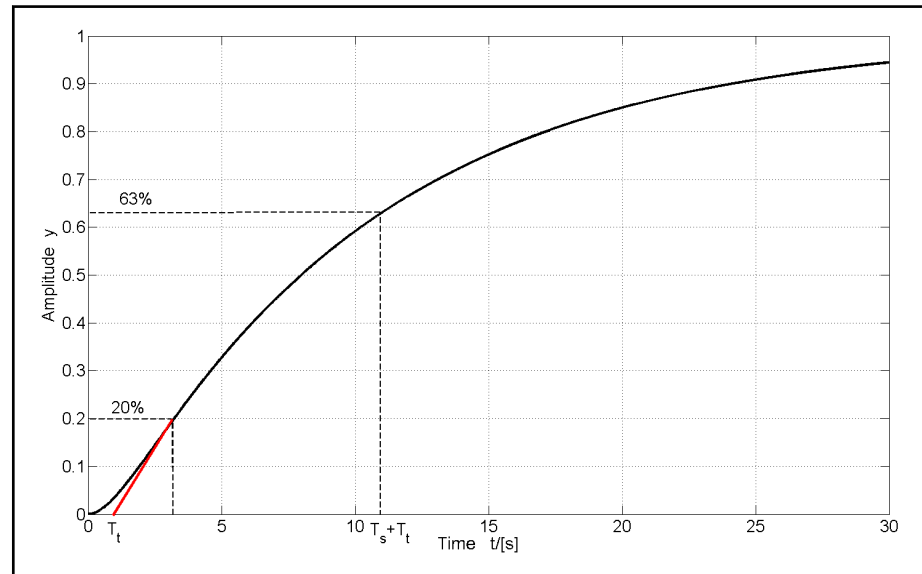


Fig. 7-108: Jump response of a non-oscillating PT_2 controlled system

In a self-adjustment, the control loop is directly closed with a P-controller with a rising edge at the "StartAutoTune" input to cause a controlled system reaction. The initial, proportional controller parameter K_P is used either from the "ContrParam" structure, calculated from the present controlled system parameters (K_s, T_s, T_t) or pre-assigned with a positive and internally stored value. While the actual value approximates the command value, the controlled system response is continuously identified and the controller settings follow the controlled system information. After the identification, the "AutoTuning" is automatically completed and the determined controller and controlled system

	parameters are applied to the "ModelParam" (K_s, T_s) and "ContrParam" (K_P, K_I, K_D, T_t) structures.
Cooling	If there is a cooling, the K_P input has to be pre-assigned with a negative value for a first system initiation. Due to the control signal limitation and the controller adaptation, $K_P = -10000$ can be specified for the initial identification.
Control signal limitation	If the actual value is changed by 20 % of the command value change, there is already sufficient information on the dynamic controlled system response to calculate the controller parameters. Thus, at this point in time, the control signal limitation "AutoTuneMaxPower" of the automatically set PI-controller is increased to the maximum controller output "OutputPowerMax" to achieve the fastest correction time possible.
Preset; controller parameter	The internal PID-controller can also be used as P-controller, PI-controller, PD-controller or PID-controller if individual controller parts are set to zero. The controller parameterization as purely integrating I-controller is not supported with regard to the expected controlled system response.
	$K_P = \text{ContrParam}.K_P, \quad K_I = \text{ContrParam}.K_I, \quad K_D = \text{ContrParam}.K_D$
	<i>Fig. 7-109: Formal dependency of the controller parameters at a "Preset"</i> In addition to the controller parameterization using the specified controller parameters, it can also be parameterized using specified controlled system parameters. If all controller parameters (K_P, K_I and K_D) of the "ContrParam" structure are zero when there is a rising edge at the "Preset" input, it is checked internally whether all controlled system parameters (K_s, T_s and T_t) are present in the "ModelParam" and "ContrParam" structures. After the plausibility check of these controlled system parameters, a PI-controller is designed on this basis. The advantage of this controller parameterization is that the set controller can be switched between the disturbance and command value response and that the controller can be re-adjusted using the "DampingFactor".
Preset; controlled system parameter	$K_P = f(\text{ModelParam}.K_s, \text{ModelParam}.T_s, \text{ModelParam}.T_t, \text{DampingFactor}, \text{ControlMode})$ $K_I = f(\text{ModelParam}.K_s, \text{ModelParam}.T_s, \text{ModelParam}.T_t, \text{DampingFactor}, \text{ControlMode})$ <i>Fig. 7-110: Formal dependency of the controller parameters at a "Preset" with controlled system parameters</i>
	 To ensure known controlled system and/or controller parameters, it is recommended to back them up in the retain area. The structures "ContrParam" and/or "ModelParam", which contain the controller and/or controlled system parameters, can be created as retain variables (VAR RETAIN).
AutoTuneMaxPower	Due to the dominant time constant T_s and a secondary delay time T_t , a jump response without control signal limitation results up to the point in time $t \sim T_s/3$. An IT_1 system can also write to the jump response (see fig. 7-108 "Jump response of a non-oscillating PT_2 controlled system" on page 480). To determine controller parameters at this early stage of the self-adjustment (up to a change of 20% of the actual value), it might be required to limit the controller output signal additionally ("AutoTuneMaxPower"). Whether this is required depends on the relation of the controlled system constants. Thus, the transient of the delay time T_t can already be completed at this point in time (20% of the actual value change) even at a low quotient of the time constants $T_s/T_t < 10$. This results in sufficient information for a controller design.

DampingFactor This input can be used to re-adjust the determined controller parameters either automatically via "StartAutoTune" or manually via "Preset" using the specification of the controlled system parameters K_s , T_s , T_i). Changing and deviating parameters not covered in a fast and slow control-loop response for ideal controlled system parameters, can thus also be considered. Therefore, the linear interpolation of the controller parameters between an oscillating and aperiodic transient was extended using an extrapolation.

For ideal controlled system parameters, a "DampingFactor" of 100 corresponds exactly to an aperiodic boundary case and a "DampingFactor" of 120 corresponds to an transient oscillation with approximately 20% overshoot. Thus, the control loop response can be decelerated accordingly with a "DampingFactor" of up to 20 or accelerated with a "DampingFactor" of up to 200. If there are controlled system parameters or a system response deviating from the real behavior, the desired dynamic control loop response can be corrected via this input.

StartUpTemp/StartUpTime If a process prior to the actual temperature control requires a pause at a certain temperature for a certain period (startup ramp), this function is available via the inputs "StartUpTemp" and "StartUpTime".

First, the controller controls up to the startup temperature "StartUpTemp". If this temperature is not reached, the startup time "StartUpTime" elapses and the controller keeps the startup temperature constant. After the startup time "StartUpTime" has elapsed, the startup ramp is completed and the controller shows its normal behavior. The startup ramp is executed with each rising edge at the "Enable" input.

Limits Alarm and status limits can be created for process monitoring purposes.

The "Limits" structure contains the values "UpperAlarm" and "LowerAlarm". When exceeding the alarm limits, an error is generated. The monitoring function is only enabled, when the current process temperature ranges between "UpperAlarm" and "LowerAlarm" in order not to generate an error during process start. These values are absolute values.

The structure "Limits" also contains the values "UpperAtSetPoint" and "LowerAtSetPoint". The "SetPointReached" is set as soon as the condition $(\text{Setpoint} - \text{Limits.LowerAtSetPoint}) \leq \text{Feedback} \leq (\text{Setpoint} + \text{Limits.UpperAtSetPoint})$ is met.

These values are relative values that relate to the "Setpoint". The "Setpoint" can thus be modified without changing "Limits.LowerAtSetPoint" and "Limits.UpperAtSetPoint". Based on this status display, it can be determined whether the desired temperature has already been reached.

Pause The closed-loop control can be paused via the "Pause" input. The control variable remains constant and the controller as well as all its values are "frozen". After the "Pause" input is reset, the closed-loop control continues. A jump can occur if the actual value has significantly changed in the meantime.

SetPointGradient The function block can limit the increase of the command value. This reduces the overshoot width and prevents the system from damages. For each new specification of the "SetPoint" input, the command value is internally changed in ramps (not in jumps) until the desired command value is reached.

"SetPointGradient" is entered in [rps]

ControlMode This input can be used to select between a control in the open control loop and a closed-loop control (optimized to the command value or disturbance response) in a closed control loop.

The manual control variable "ManContrValue" can for example be used before a self-adjustment to control the controlled variable in the operating point or when an error occurs in the actual value detection.

In the default setting `ControlMode:= CONTROL_TEMPMODE_ONLY_DISTURBANCE`, a fast as well as a robust distance identification are implemented. Observe, that it is explicitly optimized with regard to disturbance behavior in this mode (due to optimizations). Accordingly, it cannot be changed to another optimization method without another auto-tuning.

Furthermore, a failed identification due to an insufficient distance reaction, is signaled in this mode by the controller and distance parameters `NULL`.

Control loop monitoring

The function block is provided with a gradient monitoring of the actual value reporting in case of very high deviations. There can be deviations for example if the sensor displays an error code and if this error code is not caught/evaluated by the user. The gradient is set to 10 rms. A one-time exceedance is tolerated before an error is generated.

Task interval/PWM

The pulse-width modulation (PWM) is based on the task interval, i. e. the task interval is the smallest possible time slice which can be set to "ON" or "OFF" during the PWM. One PWM period is 20 times longer than the task interval. Thus, control variables can be modulated in steps of 5 % with a resolution between 0 % to 100 %.

Set the task interval/cycle time with regard to the control element and the controlled system to ensure that the PWM does not cause any visible temperature changes in one PWM period of (20*task interval). Thus, if the temperature gradient changes noticeably between the switched-on and switched-off state of the PWM period due to the control variable, the selected task interval is too high.

Do not use a task interval that is too small, since the control element (such as a relay module) is then subject to an increased load.

Furthermore, the sampling time is monitored function block-internally. This sampling time generates an error if it exceeds/falls below by more than 20 % and sets the "Error" output to TRUE.

Self-adjustment

Note the following for an optimum self-adjustment:

1. Temperature-controlled systems are mostly not linear over a wide range. Thus, if there is an identical temperature change, the following applies: The higher the temperature, the higher the control signal change required and vice versa. That results in temperature-dependent settings required by the controller. Thus, the identification in the temperature range is performed in the controller at a later stage.
2. At runtime, the self-adjustment can be started again at a rising edge at the "StartAutoTune" input. This can for example be reasonable for a level change or if there are any other modifications at the controlled system. Ensure a sufficient system initiation (actual value change) for an optimum identification.
3. The self-adjustment can only be made if the command value is higher than the actual temperature for a heating and if the actual temperature is higher than the command value for a cooling.

Special influencing variables of the self-adjustment

In special cases, the following measures can be taken to improve the "Auto-Tuning":

1. The "ExtendedSettings.IdentAccuracy" can influence the abort criteria. It is aborted either when reaching this change rate or at double the correction time with this limit. This depends on which criterion is met first.

2. Use the "ExtendedSettings.OblivionFactor" input to observe changeable controlled system parameters at the "AutoTuning". For controlled system parameters varying in time, a reduction of this input is reasonable. For controlled system parameters not varying in time, an increase of this input is reasonable.
3. The "ExtendedSettings.MinDelayTime" input specifies a minimum delay time T_t as an a priori information. This value is used for the "AutoTuning" if no value is specified at the "ContrParam.Tt" input. This value corresponds to the minimum controlled system delay time possible which cannot be lower than the sum of all secondary transport, delay and filter time constants.

Troubleshooting The function block generates the following error messages in Additional1/ Additional2 for the table F_RELATED_TABLE, 16#0170 :

ErrorID	Additional1	Additional2	Description
STATE_MACHINE_ERROR	16#00001600	Current state	Error in the state machine; invalid state
INPUT_RANGE_ERROR	16#00001601	16#00001601	Input values for maximum output value ("OutputPowerMax") outside the valid range. Value has to be higher than 0 [%] up to max. 100 [%]
INPUT_RANGE_ERROR	16#00001601	16#00001602	Input values invalid for alarm limits. (Condition: Limits.UpperAlarm > Limits.LowerAlarm)
INPUT_RANGE_ERROR	16#00001601	16#00001603	Input value for manual control variable ("ManContrValue") invalid. Value has to be between 0[%] and 100[%]
INPUT_RANGE_ERROR	16#00001601	16#00000004	Controller parameter invalid. There are either no controller parameters or the controller parameters are invalid
INPUT_RANGE_ERROR	16#00001601	16#00000005	Controlled system constant is invalid. Either the time constant "ModelParam.Ts" or "ContrParam.Tt" is invalid
INPUT_RANGE_ERROR	16#00001601	16#00000006	Input value "OblivionFactor" is invalid. Value has to be between 0.9 and 1
INPUT_RANGE_ERROR	16#00001601	16#00000007	Input value "MinDelayTime" is invalid. The value has to be higher than T#0s
INPUT_RANGE_ERROR	16#00001601	16#00000008	Input value "IdentAccuracy" is invalid. Value has to be higher than 0 [%/s]
INPUT_RANGE_ERROR	16#00001601	16#00000009	Input value "DampingFactor" is invalid. Value has to be between 20 and 200
INPUT_RANGE_ERROR	16#00001601	16#0000000B	Input value "SetPointGradient" is invalid. Value has to be higher than 0 [u/s]
INPUT_RANGE_ERROR	16#00001601	16#0000000C	Input values invalid for status limits (Condition: (Setpoint + Limits.UpperAtSetpoint) > (Setpoint - Limits.LowerAtSetpoint))
INPUT_RANGE_ERROR	16#00001601	16#00000010	Input value "ContrMode" is invalid
INPUT_RANGE_ERROR	16#00001601	16#00000011	Input value "AutoTuneMaxPower" is invalid. Value has to be between 0[%] and 100[%]

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR	16#00001601	16#00000012	Input value "ExtendedSettings.ControlType" is invalid.
INPUT_RANGE_ERROR	16#00001601	16#00000013	Input value "ExtendedSettings.bControl" is invalid. Value has to be between 0[-] and 1[-]
INPUT_RANGE_ERROR	16#00001601	16#00000014	Input value "ExtendedSettings.cControl" is invalid. Value has to be between 0[-] and 1[-]
SYSTEM_ERROR	16#00001602	Sampling time	Jittering the sampling time > 20[%]. Error becomes only active after two successive timeouts
SYSTEM_ERROR	16#00001602	16#00000005	Gradient of the actual value is too high; maybe due to a sensor error
CALCULATION_ERROR	16#00001603	16#00000007	Identified controlled system parameters are invalid, self-identification of the controlled system failed
DEVICE_ERROR	16#00001606	Temperature	Temperature error limit exceeded; temperature not in the error limits

Tab. 7-168: Error codes of the IL_TempControlType02 function block

Information on specific errors

- **"Controller parameter invalid"**

(Additional1 = 16#1601; Additional2 = 16#0004):

Cause: Either there are no controller parameters or the controller parameters created in the input/output structure "ContrParam" are incorrect. To ensure that the temperature controller can create a reasonable control signal either in the operation mode `CONTROL_TEMPMODE_OPT_COMMAND` or `CONTROL_TEMPMODE_OPT_DISTURBANCE`, at least one proportional controller with a Kp unequal to zero is required. Its number depends on the control direction (positive = heating and negative = cooling).

Error recovery: The temperature controller can either determine the controlled system and controller parameters itself, the controller parameters can be specified or the controller can determine the controller parameters itself using the controlled system parameters. For the self-determination without specified controller parameters, it is required to start the self-adjustment with a rising edge at "StartAutoTune". To specify the controller parameters, it is required to use the desired controller parameters at the input/output structure "ContrParam" and to apply them with a rising edge at the "Preset" input. To specify the controlled system parameters, it is required to set the proportional controller parameter ("ContrParam.KP") to zero in the input/output structures "ModelParam" and "ContrParam", to use the controlled system parameters ("ModelParam.Ks", "ModelParam.Ts" and "ContrParam.Tt") and to start the controller calculation with a rising edge at the "Preset" input

- **"Identified controlled system parameters are invalid, self-identification of the controlled system failed"**

(Additional1 = 16#1603; Additional2 = 16#0007):

The identification of the controlled system response and the resulting controller parameterization failed.

1. "ControlMode:= `CONTROL_TEMPMODE_ONLY_DISTURBANCE`":

Cause: The "AutoTuneMaxPower" input is too large for the existing controlled system. Consequently, the distinctive controlled system behavior could not be detected within the first half of the actual value change.

Error recovery:Reduction of the input "AutoTuneMaxPower".

2. "ControlMode:= CONTROL_TEMPMODE_OPT_XXXX":

Cause: Identification failure can cause a unfavorable specified controlled system delay time "ContrParam.Tt", a task interval that is too high, a controlled system that is not sufficiently transient at the identification point in time, a controlled system initiation that is too low or an incorrect actual value detection (sensor not connected).

Error recovery:If the self-adjustment failed due to an incorrect sampling time, a rough estimation of the controlled system delay time can be specified as information for the self-adjustment at the "ContrParam.Tt" input and/or the task interval can be reduced. If the self-adjustment of the temperature controller failed due to the controlled system not being sufficiently transient, the self-adjustment is to be repeated with a stationary transient controlled system. If the self-adjustment failed due to an insufficient controlled system initiation, repeat the self-adjustment with a bigger command value change. If the temperature detection is defective, the error has to be recovered

7.9.4 IL_TEMPCONTROL_CONTR_PARAM

The structure IL_TEMPCONTROL_CONTR_PARAM contains the current parameters K_P , K_I and K_D of the controller as well as the dead time T_t

Name	Type	Min. value	Max. value	Default value	Effective	Description
KP	REAL	n.def	n.def	10000	At a rising edge at "Pre-set" or "StartAutoTune"	Proportional gain of the controller
KI	REAL	n.def	n.def	0	At a rising edge at "Pre-set"	Integration constant of the controller
KD	REAL	n.def	n.def	0	At a rising edge at "Pre-set"	Differentiation constant of the controller
Tt	TIME	T#0s	n.def	T#0s	At a rising edge at "Pre-set", "StartAutoTune" or a change in the controller setting ("DampingFactor", "ControlMode")	Delay of the controlled system

Tab. 7-169: Structural description of the IL_TEMPCONTROL_CONTR_PARAM

7.9.5 IL_TEMPCONTROL_CONTROL_MODE

The enumeration type IL_TEMPCONTROL_CONTROL_MODE selects the control loop response:

Either the open-loop control in the open control loop or the closed-loop control optimize the command value response or the disturbance response in the closed control loop

Enumeration element	Number	Description
CONTROL_TEMP-MODE_MANUAL	1	The function block controls the actual controlled system value using the value specified at the "ManContrValue" input
CONTROL_TEMP-MODE_OPT_COMMAND	2	The function block optimizes the command value response of the "Feedback" value to the "Setpoint" value
CONTROL_TEMP-MODE_OPT_DISTURBANCE	3	The function block optimizes the disturbance response of the "Feedback" value to the "Setpoint" value
CONTROL_TEMP-MODE_ONLY_DISTURBANCE	4	The function block optimizes the disturbance response of the "Feedback" value to the "Setpoint" value Fast and robust auto-tuning explicitly for interference behavior at the same time. (Default)

Tab. 7-170: IL_TEMPCONTROL_CONTROL_MODE enumeration type

7.9.6 IL_TEMPCONTROL_CONTROL_TYPE

The enumeration type IL_TEMPCONTROL_CONTROL_TYPE is used to select a specific controller type (PI or PID) used as basis for the internal controller parameterization ("Autotune" or "Preset") with the distance parameters.

Depending on this "auto-tuning" setting, the internal PID controller can be parameterized as P-, PI- or PID controller via a "preset" of the respective controller parameters.

Enumeration element	Number	Description
CONTROL_TYPE_PI	1	Parameterization of a PI-controller using the controlled system parameters
CONTROL_TYPE_PID	2	Parameterization of a PID-controller using the controlled system parameters

Tab. 7-171: IL_TEMPCONTROL_CONTROL_TYPE enumeration type

7.9.7 IL_TEMPCONTROL_EXTENDED_SETTINGS

The structure IL_TEMPCONTROL_EXTENDED_SETTINGS contains the extended setting options of the self-adjustment.

The default settings are sufficient for the common controlled systems.

Name	Type	Min. value	Max. value	Default value	Effective	Description
IdentAccuracy	REAL	>0	N. def	3.0	Cyclic	Abort criterion of the identification
OblivionFactor	REAL	0.9	1.0	0.98	Cyclic	Oblivion factor for changeable controlled systems
MinDelayTime	TIME	>T#0s	N. def	T#1s	At a rising edge of "StartAutoTune"	Minimum delay of the controlled system
ControlType	IL_TEMP- CON- TROL_CON- TROL_TYPE , page 487	CON- TROL_TYPE _PI	CON- TROL_TYPE _PID	CON- TROL_TY PE_PI	Cyclic	Controller type (PI- or PID-controller)
bControl	REAL	0	1.0	1.0	Cyclic	Scaling factor for P-value
cControl	REAL	0	1.0	1.0	Cyclic	Scaling factor for D-value

Tab. 7-172: Structural description of the IL_TEMPCONTROL_EXTENDED_SETTINGS

7.9.8 IL_TEMPCONTROL_LIMITS

The structure IL_TEMPCONTROL_LIMITS contains the alarm limits in [u] (units). Initial value is 0u (not active).

If values are assigned to the structure,

Limits.UpperAlarm > Limits.LowerAlarm

and

(Setpoint + Limits.UpperAtSetpoint) > (Setpoint - Limits.LowerAtSetpoint)

Otherwise an error is generated

Name	Type	Min. value	Max. value	Default value	Effective	Description
UpperAlarm	REAL	n.def	n.def	0	Cyclic	Upper actual value limit when an alarm is triggered (absolute)
LowerAlarm	REAL	n.def	n.def	0	Cyclic	Lower actual value limit when the alarm is triggered (absolute)
UpperAtSetpoint	REAL	n.def	n.def	0	Cyclic	Upper actual limit for SetpointReached (relative)
LowerAtSetpoint	REAL	n.def	n.def	0	Cyclic	Lower actual limit for SetpointReached (relative)

Tab. 7-173: Structural description of the IL_TEMPCONTROL_LIMITS

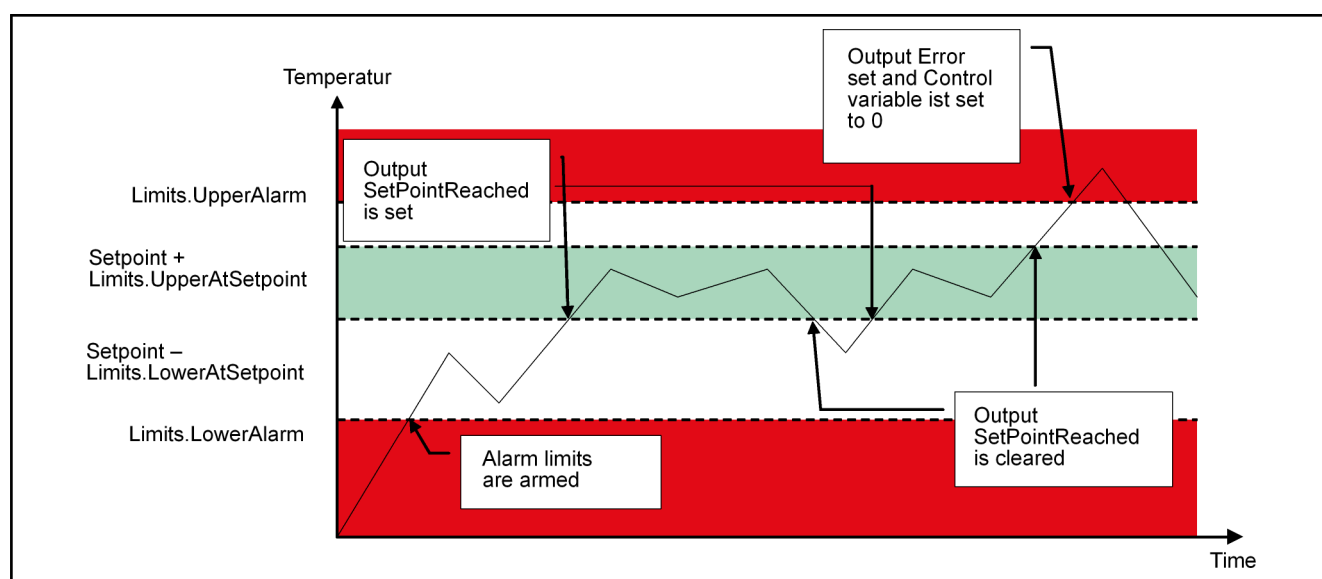


Fig. 7-111: Mode of operation of the alarm and status limits

7.9.9 IL_TEMPCONTROL_MODEL_PARAM

The structure IL_TEMPCONTROL_MODEL_PARAM contains the parameters K_S and T_S of the controlled system.

Name	Type	Min. value	Max. value	Default value	Effective	Description
K_S	REAL	n.def	n.def	0	At a rising edge at "Preset", "StartAutoTune" or a change in the controller setting ("DampingFactor", "ControlMode")	Gain of the controlled system
T_S	TIME	T#0s	n.def	T#0s	Rising edge at "Preset", "StartAutoTune" or a change in the controller setting ("DampingFactor", "ControlMode")	Compensation time constant of the controlled system

Tab. 7-174: Structural description of the IL_TEMPCONTROL_MODEL_PARAM

7.10 Two-position controller

7.10.1 IL_TwoPosCtrlType01

Brief description The function block IL_TwoPosCtrlType01 implements a two-position controller with hysteresis and is suitable for simple closed-loop controls.



In 14VRS, the "ML_TwoPosCtrlType01 (ML_TechBase library)" was renamed and moved to the "RIL_LoopControl library".

Assignment: Target system/library

Target system	Library
MLC 15VRS	RIL_LoopControl

Tab. 7-175: Reference table of IL_TwoPosCtrlType01 function block

Interface description

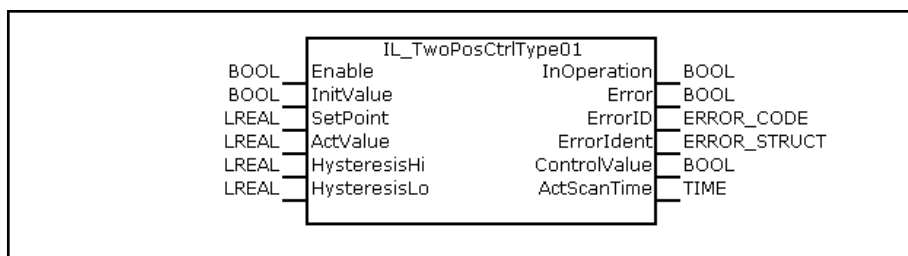


Fig. 7-112: IL_TwoPosCtrlType01 function block

I/O type	Name	Data type	Description
VAR_INPUT	Enable	BOOL	Processing of function block enabled (cyclic, state-controlled)
	InitValue	BOOL	Specified value for the control variable y
	SetPoint	LREAL	Reference variable w (command value)
	ActValue	LREAL	Controlled variable x (actual value)
	HysteresisHi	LREAL	Upper response threshold of the hysteresis
	HysteresisLo	LREAL	Lower response threshold of the hysteresis
VAR_OUTPUT	InOperation	BOOL	Calculation of the control variable y completed without error, output variable "ControlValue" valid
	Error	BOOL	Calculation of the control variable "ControlValue" completed with error, output variable "ErrorIdent" valid
	ErrorID	ERROR_CODE	If the "Error" output is set, it contains a broad error classification
	ErrorIdent	ERROR_STRUCT	Diagnostic description in the event of an error Troubleshooting, page 491
	ControlValue	BOOL	Controller output value - control variable y
	ActScanTime	TIME	Controller cycle time [ms]

Tab. 7-176: Interface variables of the IL_TwoPosCtrlType01 function block

Specification

The function block IL_TwoPosCtrlType01 can be adjusted in its properties using the following input variables:

"HysteresisHi":

- The input has to be preset with a positive value or zero. Zero switches off the positive hysteresis.

"HysteresisLo":

- The input has to be preset with a negative value or zero. Zero switches off the negative hysteresis.



When calculating the control variable "ControlValue", "FALSE" is dominant. Thus, with "HysteresisLo" == "HysteresisHi" == zero with a control difference of zero ("SetPoint" == "ActualValue"), the output "ControlValue" is always "FALSE".

Functional description

After the function block has been enabled for processing, the function block IL_TwoPosCtrlType01 uses "Enable" to preset the internal controller state in the first controller cycle as well as the controller output to the preset value given at the "InitValue" input and it cyclically calculates the control variable "ControlValue" in the following cycles.

If there are no errors during the function block processing, the validity of the control variable is signaled by the "InOperation" output and the elements of the output structures "ErrorID" and "ErrorIdent" are not updated.

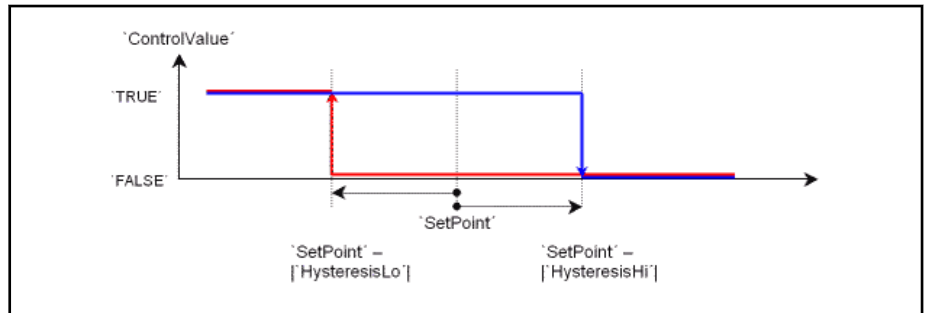


Fig. 7-113: Hysteresis of the IL_TwoPosCtrlType01 function block

If an error occurs during the function block processing, this is signaled by the "Error" output and specified by the elements of the output structures "ErrorID" and "ErrorIdent" which are updated in this case. The outputs "ControlValue" and "ActScanTime" are not updated in the event of an error.

Troubleshooting

The function block uses the error table **MLC_TABLE**, 16#0030. It can generate the following error messages in Additional1/Additional2.

ErrorID	Additional1	Additional2	Description
INPUT_RANGE_ERROR (16#0006)	16#F0260030	0	The inputs "HysteresisHi" or "HysteresisLo" are invalid
STATE_MACHINE_ERROR (16#0005)	16#F0260031	xx	Invalid state of the state machine xx indicates the invalid state

Tab. 7-177: Error codes of the IL_TwoPosCtrlType01 function block

7.11 Utilities

7.11.1 IL_COORD_TYPE01

This data structure defines a two-dimensional coordinate with X and Y REAL values.

Program:

```

TYPE IL_COORD_TYPE01 :
STRUCT
  X: REAL;    // X coord
  Y: REAL;    // Y coord
END_STRUCT
END_TYPE

```

Usage in

- [IL_DeviationResponseCurve](#), page 360,
- [IL_PType01](#), page 346,
- [IL_PType02](#), page 350,

7.11.2 IL_SUPPRESSION_LIMITS

IL_SUPPRESSION_LIMITS is an enumeration type and enables the suppression window with the following allocation:

Element	Description
IL_SUPPRESSION_OFF	Suppression is disabled and all values remain unchanged at the output.
IL_SUPPRESSION_LIMIT_1	Suppression window #1 active ("SuppressionLimit1Low" and "SuppressionLimit1High")
IL_SUPPRESSION_LIMIT_2	Suppression window #2 active ("SuppressionLimit2Low" and "SuppressionLimit2High")

Tab. 7-178: Structure of the enumeration type *IL_SUPPRESSION_LIMITS*

Usage in

- [IL_SuppressionType03](#), page 447
- [IL_SuppressionType04](#), page 452

7.11.3 IL_SUPPRESSION_MODE

IL_SUPPRESSION_MODE is an enumeration type and describes the function of the inputs "SuppressionLimitLow*" and "SuppressionLimitHigh*" with the following allocation:

Element	Description
IL_SUPPRESSION_PERCENTAGE	Values are on a percentage rate to the averaged input value
IL_SUPPRESSION_ABSOLUTE_VALUE	Values are absolute values to the averaged input value
IL_SUPPRESSION_STATISTICAL	Value are statistical to the default deviation σ of the input value (e.g. $0.5 \cdot \sigma$, $1.0 \cdot \sigma$, $4.0 \cdot \sigma$,...)

Tab. 7-179: Structure of the enumeration type *IL_SUPPRESSION_MODE*

Usage in

- [IL_SuppressionType01](#), page 438
- [IL_SuppressionType02](#), page 442
- [IL_SuppressionType03](#), page 447
- [IL_SuppressionType04](#), page 452

8 RIL_HMI_Uilities.library

8.1 Introduction and overview



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

Areas of application and functions/function blocks of RIL_HMI_Uilities

Area of application (folder in library)	
Function blocks	Description
POUs	
IL_MKeys, page 493	Supports the operation of machine function keys at the visualization device
IL_UdpKeys, page 494	The user can safely apply "pressing the buttons" from an HMI device to the control

Tab. 8-1: RIL_HMI_Uilities library

8.2 IL_MKeys

Brief description

The function block IL_MKeys supports the operation of machine function keys on the visualization device, initiating the execution of machine functions by the PLC.

If available as PLC inputs, wire machine function keys to the inputs R1-T8 and L1-L8.

Target system	Library/folder
MLC 15VRS Basic function package PLC	RIL_HMI_Uilities/POUs

Tab. 8-2: Library assignment

Interface description

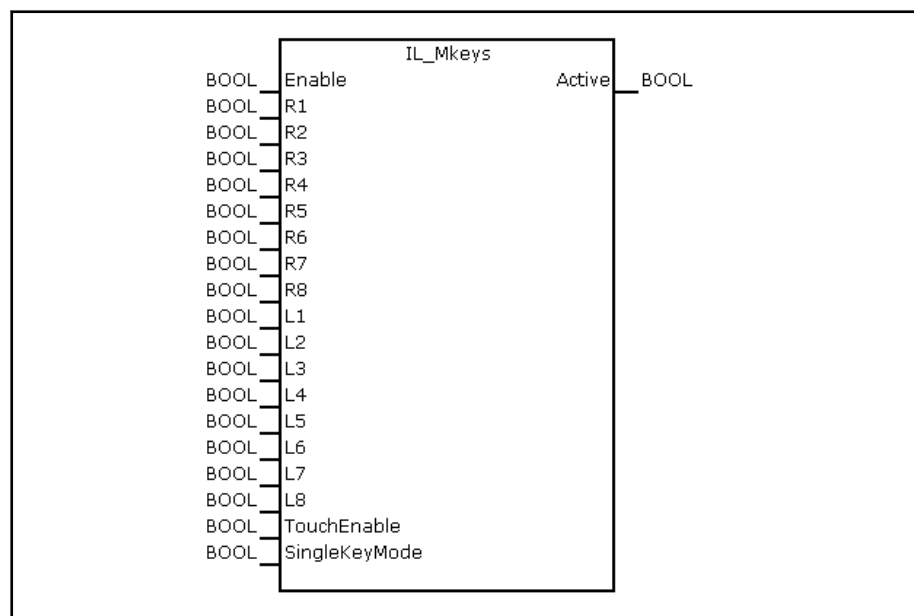


Fig. 8-1: Function block IL_MKeys

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	Activates the function block.
	R1-R8	BOOL	Hardware keys <R1> to <R8>.
	L1-L8	BOOL	Hardware keys <L1> to <L8>.
	TouchEnable	BOOL	Allows operation via touch screen or click on the keys at the visualization device.
	SingleKeyMode	BOOL	Operation mode ensuring that only one key can be pressed at a time
VAR_OUTPUT	Active	BOOL	Is TRUE if an M-key assignment is active.

Tab. 8-3: Function block IL_MKeys

⚠ CAUTION**Danger with safety-critical machine functions!**

Since it is not ensured that releasing touch keys is detected by the PLC, assign a hardware enabling button to the "TouchEnable" input where safety-critical machine functions are concerned.



Due to compatibility reasons, the RIL_HMI_Uilities.library also contains the function blocks MKEYS and HMICE_MKEYS each available in IndraLogic 1x.

In case of new implementations in IndraLogic 2G we recommend not to use these function blocks anymore.

8.3 IL_UdpKeys

Brief description

The function block IL_UdpKeys allows the user to reliably transfer keystrokes of an HMI device to the control.

Connection via UDP. Thus, the function block only operates when the PC or the embedded PC support the UDP protocol (e.g. VEH30.2).

Every time the function block is called, the last data is copied.

In case of a connection abort, all outputs are reset to 0. This allows to program safe properties, with the safety mode being assigned to 0 in the output data of the function block.

Target system	Library/folder
MLC 15VRS	RIL_HMI_Uilities/POUs
Basic function package PLC	

Tab. 8-4: Library assignment

Interface description

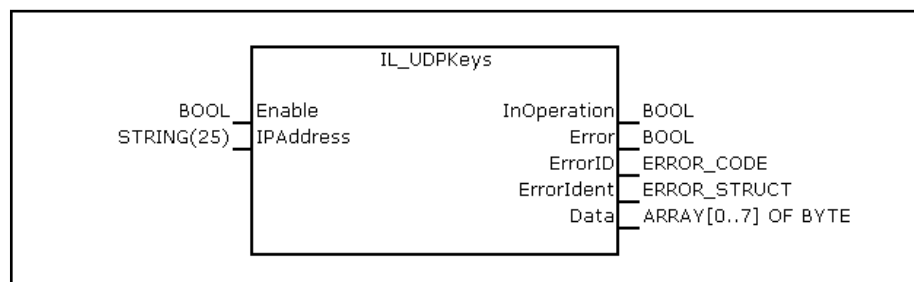


Fig. 8-2: Function block IL_UdpKeys

	Name	Type	Comment
VAR_INPUT	IPAddress	STRING(25)	Entered IP address "." notation, e.g. "10.104.73.193"
	Enable	BOOL	If the input is TRUE, the data is permanently refreshed. If inactive, all output data is 0.
VAR_OUTPUT	InOperation	BOOL	If TRUE, the output data is valid and refreshed.
	Error	BOOL	Indicates that an error occurred in the function block. The output data is set to 0.
	ErrorID	ERROR_CODE	Error detection (see Error messages, page 495).
	ErrorIdent	ERROR_STRUCT	Error structure with further error subdivision.
	Data	ARRAY [0..7] OF BYTE	Contains data as byte array. The meaning of the respective bits is described in the HMI documentation.

Tab. 8-5: Function block IL_UdpKeys

Installation note

To use the function block, enable the UDP protocol on the HMI device.

For details, refer to the project planning manual of the respective hand-held terminal.

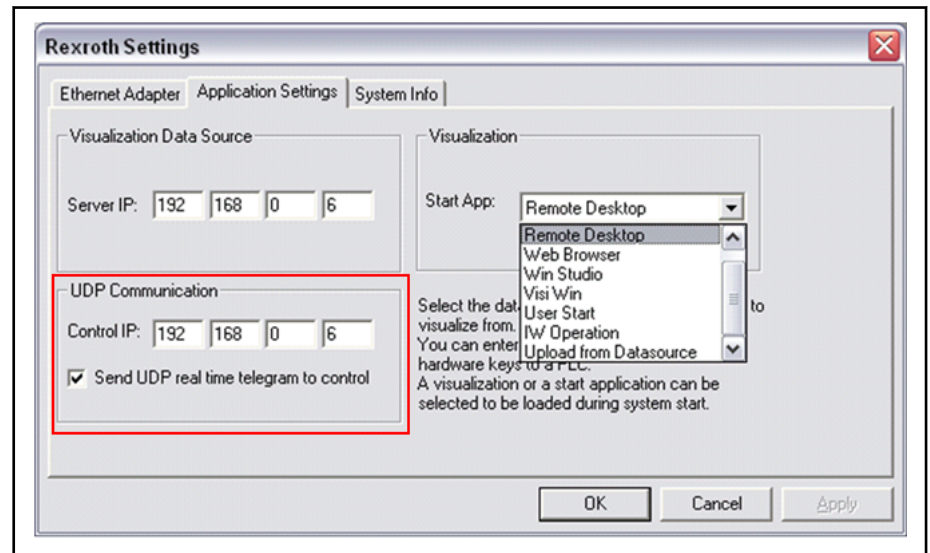


Fig. 8-3: Rexroth settings, setting the UDP communication to transfer keyboard code

Error messages

The function block generates the following error messages in Additional1 and Additional2 for table **ETHERNET_TABLE, 16#0150**:

ErrorID	Additional1	Additional2	Description
INPUT_INVALID_ERROR (16#0001)	10	0	Wrong IP syntax
COMMUNICATION_ERROR (16#0002)	11	0	Unknown IP address or no connection
DEVICE_ERROR (16#0008)	12	0	Invalid internal buffer size
STATE_MACHINE_ERROR (16#0005)	1	0	Invalid state of the state machine

Tab. 8-6: Generated error codes of the IL_UdpKeys function block



For compatibility reasons, the RIL_HMI_Utility.library also contains function block IL_VExKeys available in IndraLogic 1x.

In case of new implementations in IndraLogic 2G, we recommend not to use this function block anymore.

9 IL_VCP_DP.library

9.1 Introduction and overview



The basic function package PLC is sufficient for the library of the MLC 15VRS system.

This library is used to couple Rexroth VCP small operator terminals with Profibus connection to the PLC. Optionally, three function blocks can be used:

- [VCP_PBS16_A4096](#) , page 497
- [VCP_PBS32_A4096](#) , page 498
- [VCP_PBS32_A65536](#) , page 499

9.2 VCP_PBS16_A4096

Brief description

This function block enables the Profibus-DP protocol for the small operator terminals VCPxx. Additionally, the I/O image of the physical addresses between PLC and operator terminal is transferred.

Data width of data transfer is 16 bytes. The size of the address range available via an ARRAY is 4096 bytes (total inputs and outputs).

Interface description

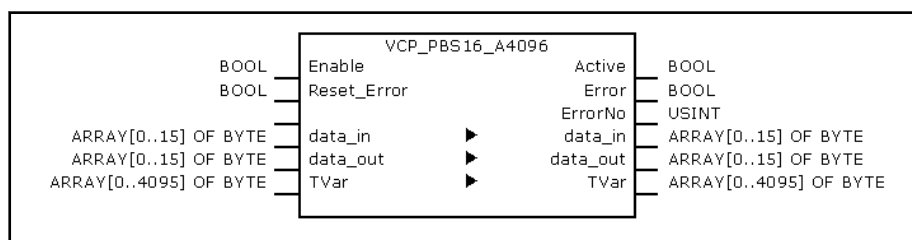


Fig. 9-1: VCP_PBS16_A4096

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	TRUE: Function block is processed FALSE: Function block is not processed
	Reset_Error	BOOL	TRUE: Resetting "Error" (to FALSE) and "ErrorNo" is set to 0
VAR_OUTPUT	Active	BOOL	TRUE if "Enable" is also TRUE
	Error	BOOL	TRUE when an error occurs. Reset with "Reset-Error".
	ErrorNo	USINT	Error type: 2: InputRangeError 4: CalculationError
VAR_IN_OUT	data_in	ARRAY [0..15] OF BYTE	Data to couple the physical inputs of the small operator panel
	data_out	ARRAY [0..15] OF BYTE	Data to couple the physical outputs of the small operator panel
	TVar	ARRAY [0..4095] OF BYTE	Array to read from and write to the small operator panel.

Tab. 9-1: VCP_PBS16_A4096 interface

- VI-Composer** When configuring with the Rexroth VI-Composer, the addresses of the variable list refer to the respective byte in the "TVar" array used to exchange data in the PLC program.
- Troubleshooting** When an error occurs, the communication is canceled and the display of the small operator panel outputs COMMUNICATION ERROR, ERROR CODE 110.
- The error type (ErrorNo) indicates whether it is an address calculation error (CalculationError) or whether the data width selected for "Data_in or "Data_out" was too wide (InputRangeError).

9.3 VCP_PBS32_A4096

Brief description This function block enables the Profibus-DP protocol for the small operator terminals VCPxx. Additionally, the I/O image of the physical addresses between PLC and operator terminal is transferred.

Data width of data transfer is 32 bytes. The size of the address range available via an ARRAY is 4096 bytes (total inputs and outputs).

Interface description

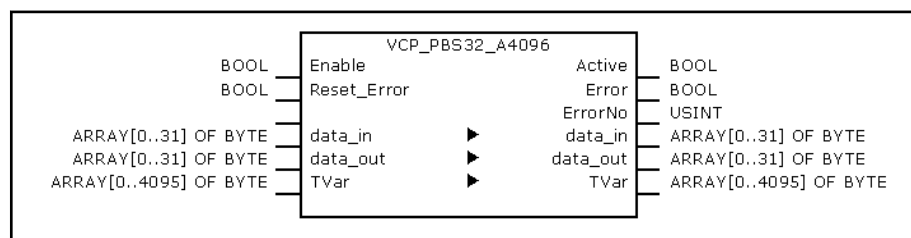


Fig. 9-2: VCP_PBS32_A4096

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	TRUE: Function block is processed FALSE: Function block is not processed
	Reset_Error	BOOL	TRUE: Resetting "Error" (to FALSE) and "ErrorNo" is set to 0
VAR_OUTPUT	Active	BOOL	TRUE if "Enable" is also TRUE
	Error	BOOL	TRUE when an error occurs. Reset with "Reset-Error".
	ErrorNo	USINT	Error type: 2: InputRangeError 4: CalculationError
VAR_IN_OUT	Data_in	ARRAY [0..31] OF BYTE	Data to couple the physical inputs of the small operator panel
	Data_out	ARRAY [0..31] OF BYTE	Data to couple the physical outputs of the small operator panel
	TVar	ARRAY [0..4095] OF BYTE	Array to read from and write to the small operator panel.

Tab. 9-2: VCP_PBS32_A4096 interface

VI-Composer When configuring with the Rexroth VI-Composer, the addresses of the variable list refer to the respective byte in the "TVar" array used to exchange data in the PLC program.

Troubleshooting When an error occurs, the communication is canceled and the display of the small operator panel outputs COMMUNICATION ERROR, ERROR CODE 110.

The error type (ErrorNo) indicates an address calculation error (CalculationError).

9.4 VCP_PBS32_A65536

Brief description This function block enables the Profibus-DP protocol for the small operator terminals VCPxx. Additionally, the I/O image of the physical addresses between PLC and operator terminal is transferred.

Data width of data transfer is 32 bytes. The size of the address range available via an ARRAY is 65536 bytes (total inputs and outputs).

Interface description

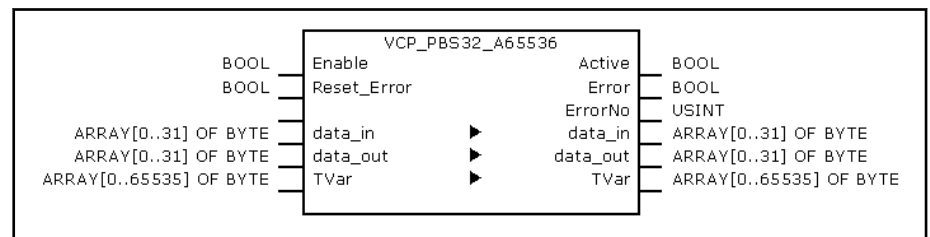


Fig. 9-3: VCP_PBS32_A65536

	Name	Type	Comment
VAR_INPUT	Enable	BOOL	TRUE: Function block is processed FALSE: Function block is not processed
	Reset_Error	BOOL	TRUE: Resetting "Error" (to FALSE) and "ErrorNo" is set to 0
VAR_OUTPUT	Active	BOOL	TRUE if "Enable" is also TRUE
	Error	BOOL	TRUE when an error occurs. Reset with "Reset-Error".
	ErrorNo	USINT	Error type: 2: InputRangeError 4: CalculationError
VAR_IN_OUT	Data_in	ARRAY [0..31] OF BYTE	Data to couple the physical inputs of the small operator panel
	Data_out	ARRAY [0..31] OF BYTE	Data to couple the physical outputs of the small operator panel
	TVar	ARRAY [0..65535] OF BYTE	Array to read from and write to the small operator panel.

Tab. 9-3: VCP_PBS32_A65536 interface

VI-Composer When configuring with the Rexroth VI-Composer, the addresses of the variable list refer to the respective byte in the "TVar" array used to exchange data in the PLC program.

Troubleshooting When an error occurs, the communication is canceled and the display of the small operator panel outputs COMMUNICATION ERROR, ERROR CODE 110.

The error type (ErrorNo) indicates an address calculation error (CalculationError).

10 Service and support

Our worldwide service network provides an optimized and efficient support. Our experts offer you advice and assistance should you have any queries. You can contact us **24/7**.

Service Germany Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related queries for electric drive and controls.

Contact the **Service Hotline** and **Service Helpdesk** under:

Phone:	+49 9352 40 5060
Fax:	+49 9352 18 4941
E-mail:	service.svc@boschrexroth.de
Internet:	http://www.boschrexroth.com

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide Outside Germany, please contact your local service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information To be able to help you more quickly and efficiently, please have the following information ready:

- Detailed description of malfunction and circumstances
- Type plate specifications of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your e-mail address)

Index

A

About this documentation.....	1
Validity of the documentation.....	1
Active.....	6
Adjustment functions	
IL_ContinuousAdjustType02.....	72, 75
IL_IncrementalAdjustType01.....	78
ANSI Z535.6-2006.....	2
Asynchronous communication function blocks.....	311
Asynchronous FTP function blocks.....	295, 306
Asynchronous ICMP function blocks.....	293
AXIS_GROUP_REF.....	23
AXIS_REF.....	21

B

Basic class	
IL_EnableDoneBase.....	86
IL_EnableInOpBase.....	89, 91
IL_ExecuteDoneAbortedBase.....	100
IL_ExecuteDoneAbortedShutdownBase.....	102
IL_ExecuteDoneBase.....	94
IL_ExecutelnOpBase.....	97
Basic classes for the function block development.....	80
Basic elements - Control engineering.....	354
Basic libraries	
Overview.....	1
Big-endian.....	258
Byte order.....	257

C

Characters strings without length limitations.....	136
Checksums	
CRC16.....	124
CommandAborted.....	6
Controllers	
PI.....	330
PID.....	330
CONTROLS.....	21
Conversion	
Bit-stream connect.....	107
Bit-stream split.....	115
ByteSwap.....	123
Data type change.....	126
Cyclic communication.....	257

D

Data types for synchronization in the position-controlled synchronous operation modes....	50
Data types to diagnose errors in the Motion function blocks.....	54
Data types to reference objects of Motion function blocks.....	20

Data types to report date and absolute time specifications for Motion function blocks.....	57
Data types to support cams, MotionProfiles and FlexProfiles.....	26
Data types to support touch probe functions.....	58
Debugging.....	129
Documentation	
Change record.....	1
Done.....	6
Double-linked list.....	184
Double-linked lists.....	184

E

Enable.....	6
ENUM_INT_OPTION.....	161
ENUM_OPT_UTF.....	163
ENUM_REAL_OPTION.....	162
ENUM_TRIM_OPT.....	174
Error codes	
IL_AmplitudeControllerType01.....	425
IL_AmplitudeControllerType02.....	428
IL_AngleGeneratorType01.....	411
IL_AngleGeneratorType02.....	412
IL_ContinuousAdjustType01.....	75
IL_ContinuousAdjustType02.....	78
IL_CRC16.....	126
IL_DeadtimeType01.....	357
IL_DeadtimeType02.....	360
IL_DeviationResponseCurve.....	362
IL_DT1Type01.....	364
IL_DT1Type02.....	367
IL_FIRType01.....	389
IL_FIRType02.....	391
IL_FTPGetAsync.....	298
IL_FTPPutAsync.....	301
IL_IIRType01.....	394
IL_IIRType02.....	396
IL_IncrementalAdjustType01.....	80
IL_InterpolationLinear.....	135
IL_LinkedList.....	188
IL_LocalTime.....	178
IL_ModbusTCPClient.....	221
IL_ModbusTCPReadCoils.....	224
IL_ModbusTCPReadDeviceIdentification....	243
IL_ModbusTCPReadDiscreteInputs.....	226
IL_ModbusTCPReadHoldingRegisters.....	228
IL_ModbusTCPReadInputRegisters.....	230
IL_ModBusTCPReadWriteMultipleRegisters.....	241
IL_ModbusTCPServer.....	212
IL_ModbusTCPTType02.....	217
IL_ModBusTCPWriteMultipleCoils.....	236
IL_ModBusTCPWriteMultipleRegisters.....	238
IL_ModBusTCPWriteSingleCoil.....	232
IL_ModBusTCPWriteSingleRegister.....	234

Index

IL_NoiseGeneratorType01.....	404
IL_NoiseGeneratorType02.....	405
IL_PDT1Type01.....	370
IL_PDT1Type02.....	373
IL_PhaseController.....	414
IL_PIDType01.....	334, 350
IL_PIDType02.....	338, 354
IL_PIDType03.....	342
IL_PIDType04.....	346
IL_Ping.....	295
IL_PingAsync.....	294
IL_PRBSGeneratorType01.....	407
IL_PRBSGeneratorType02.....	410
IL_PT1Type01.....	376
IL_PT1Type02.....	378
IL_SignalGeneratorType01.....	418
IL_SignalGeneratorType02.....	422
IL_StateVariableType01.....	398
IL_StateVariableType02.....	400
IL_SuppressionType01.....	442
IL_SuppressionType02.....	447
IL_SuppressionType03.....	452
IL_SuppressionType04.....	458
IL_TCPClose.....	279
IL_TCPCloseAsync.....	273
IL_TCPConnectionAsync.....	263
IL_TCPCyclic.....	315
IL_TCPInitial.....	275
IL_TCPInitialAsync.....	267
IL_TCPRecv.....	277
IL_TCPRecvAsync.....	270
IL_TCPRecvPacket.....	279
IL_TCPRecvPacketAsync.....	272
IL_TCPSend.....	276
IL_TCPSendAsync.....	268
IL_TempControlType01.....	475
IL_TempControlType02.....	485
IL_TransportDelayType01.....	381
IL_TransportDelayType02.....	384
IL_TwoPosCtrlType01.....	491
IL_UDPClose.....	293
IL_UDPCloseAsync.....	288
IL_UDPCyclic.....	319
IL_UDPInitial.....	290
IL_UDPInitialAsync.....	284
IL_UdpKeys.....	495
IL_UDPRecvfrom.....	292
IL_UDPRecvfromAsync.....	287
IL_UDPSendto.....	291
IL_UDPSendtoAsync.....	286
IL_UDPSocketAsync.....	282
IL_ValueLimit.....	385
IL_ValueWindow.....	387
Error tables	
ETHERNET_TABLE.....	322
ERROR_CODE.....	54
ERROR_STRUCT.....	55
ERROR_TABLE.....	56

Error_Tables	
ETHERNET_TABLE.....	322
ETHERNET_TABLE.....	322
16#00000001-16#00000212.....	322
16#00001001-16#00001002.....	324
16#00002001-16#00002004.....	324
16#00002101-16#00002108.....	325
Event-controlled function blocks.....	330
Execute.....	6
ExecuteLock.....	6
EXT_SYS_TIME_DATE.....	57

F

Filter.....	387
FTP.....	256
Function block types.....	5
Function block types and their interfaces.....	6

H

Hazard warnings.....	1
Helpdesk.....	501
Hotline.....	501

I

I/O Time response of the function blocks.....	245
ICMP.....	256
IIL_IIRType01.....	391
IIL_IIRType02.....	394
IL_AMP_CONTROL.....	428
IL_AmplitudeControllerType01.....	422
IL_AmplitudeControllerType02.....	425
IL_AngleGeneratorType01.....	410
IL_AngleGeneratorType02.....	411
IL_Ascii_To_Char.....	159
IL_AsciiStreamToBase64.....	143
IL_AsciiToByte.....	126
IL_Base64ToAsciiStream.....	145
IL_BoolToByte.....	108
IL_BoolToDWord.....	109
IL_BoolToWord.....	111
IL_BufferGetElementCount.....	164
IL_BufferToString.....	147
IL_ByteToAscii.....	126
IL_ByteToBool.....	115
IL_ByteToDWord.....	112
IL_CENTERLINE_CONTROL.....	428
IL_Char_To_Ascii.....	160
IL_Code.....	165
IL_CONCAT.....	152
IL_ConcatByte.....	113
IL_ConcatWord.....	114
IL_CONN_MODE.....	320
IL_CONN_STATUS.....	321
IL_ContinuousAdjustType01.....	72
IL_ContinuousAdjustType02.....	75
IL_COORD_PAIR_TYPE01.....	136
IL_COORD_TYPE01.....	491

IL_CRC16.....	125	IL_ModbusTCPClient.....	219
IL_CURVETYPE.....	429	IL_ModbusTCPReadCoils.....	221
IL_Date.....	175	IL_ModbusTCPReadDeviceIdentification.....	241
IL_DateAndTime.....	175	IL_ModbusTCPReadDiscreteInputs.....	224
IL_DeadtimeType01.....	355	IL_ModbusTCPReadHoldingRegisters.....	226
IL_DeadtimeType02.....	357	IL_ModbusTCPReadInputRegisters.....	228
IL_DELETE.....	153	IL_ModbusTCPReadWriteMultipleRegisters....	238
IL_DeviationResponseCurve.....	360	IL_ModbusTCPServer.....	209
IL_DT1Type01.....	362	IL_ModbusTCPServerType02.....	212
IL_DT1Type02.....	364	IL_ModbusTCPWriteMultipleCoils.....	234
IL_DWordToBool.....	116	IL_ModbusTCPWriteMultipleRegisters.....	236
IL_DWordToByte.....	119	IL_ModbusTCPWriteSingleCoil.....	230
IL_EnableDoneBase.....	86	IL_ModbusTCPWriteSingleRegister.....	232
IL_EnableInOpBase.....	89	IL_NoiseGeneratorType01.....	402
IL_EnableInOpShutdownBase.....	91	IL_NoiseGeneratorType02.....	404
IL_ExecuteDoneAbortedBase.....	100	IL_NULL.....	20
IL_ExecuteDoneAbortedShutdownBase.....	102	IL_PDT1Type01.....	367
IL_ExecuteDoneBase.....	94	IL_PDT1Type02.....	370
IL_ExecuteInOpBase.....	97	IL_PhaseController.....	412
IL_ExtSysTimeDate.....	176	IL_PIDType01.....	331
IL_FB_STATES.....	107	IL_PIDType02.....	334
IL_FILTER_TYPE.....	400	IL_PIDType03.....	338
IL_FIND.....	154	IL_PIDType04.....	342
IL_FIR_COEFF.....	401	IL_Ping.....	295
IL_FIRType01.....	387	IL_PingAsync.....	293
IL_FIRType02.....	389	IL_PIType01.....	346
IL_FTPGetAsync.....	296	IL_PIType02.....	350
IL_FTPGetFileAsync.....	301	IL_PRBSGeneratorType01.....	405
IL_FTPPutAsync.....	298	IL_PRBSGeneratorType02.....	407
IL_FTPPutFileAsync.....	304	IL_PT1Type01.....	373
IL_GetModuloType01.....	190	IL_PT1Type02.....	376
IL_GetModuloType02.....	191	IL_Real_To_Str.....	161
IL_GetModuloType03.....	192	IL_RealToString.....	127
IL_GetModuloType04.....	193	IL_REPLACE.....	157
IL_HighByte.....	119	IL_RIGHT.....	158
IL_HighResTimeDiff.....	129	IL_SignalGeneratorType01.....	414
IL_HighResTimeTick.....	130	IL_SignalGeneratorType02.....	418
IL_HighWord.....	120	IL_StateVariableType01.....	396
IL_HistogramType01.....	429	IL_StateVariableType02.....	398
IL_HistogramType02.....	431	IL_StatisticType01.....	433
IL_IIR_COEFF.....	401	IL_StatisticType02.....	435
IL_IncrementalAdjustType01.....	78	IL_StrCompare.....	141
IL_INSERT.....	154	IL_StrCompareNoCase.....	141
IL_Int_To_Str.....	160	IL_StrCopy.....	150
IL_InterpolationLinear.....	133	IL_StrGetElement.....	167
IL_LEFT.....	155	IL_String_To_Unicode.....	162
IL_LEN.....	156	IL_StringToBuffer.....	148
IL_LINKED_LIST_MAINTENANCE.....	188	IL_StrInverse.....	166
IL_LINKED_LIST_MODE.....	189	IL_StrLChr.....	168
IL_LinkedList.....	184	IL_StrLower.....	139
IL_LocalTime.....	176	IL_StrNCompare.....	142
IL_LowByte.....	120	IL_StrNCompareNoCase.....	143
IL_LowWord.....	121	IL_StrNCopy.....	151
IL_LRealToString.....	127	IL_StrRChr.....	169
IL_MaxValue.....	135	IL_StrSetElement.....	170
IL_MID.....	156	IL_StrSetEndOfString.....	166
IL_MKeys.....	493	IL_StrSplit.....	171
IL_MODBUS_STATS.....	218	IL_StrTrimAll.....	172

Index

IL_StrTrimSpace.....	173	IL_UDPSendtoAsync.....	284
IL_StrUpper.....	140	IL_UDPSocketAsync.....	280
IL_SUPPRESSION_LIMITS.....	491	IL_Unicode_To_String.....	163
IL_SUPPRESSION_MODE.....	492	IL_ValueLimit.....	384
IL_SuppressionType01.....	438	IL_ValueWindow.....	385
IL_SuppressionType02.....	442	IL_VCP_DP.library.....	497
IL_SuppressionType03.....	447	IL_WordToBool.....	121
IL_SuppressionType04.....	452	InGear.....	6
IL_SwapDWord.....	123	InOperation.....	6
IL_SwapWord.....	124	Inputs and outputs of edge-controlled function blocks, final processing.....	10
IL_SysTime64.....	178	Inputs and outputs of edge-triggered function blocks, permanent processing.....	16
IL_SysTime64ToSysTimeDate.....	179	Inputs and outputs of state-controlled function blocks, final processing.....	7
IL_SysTimeDate.....	179	Inputs and outputs of state-controlled function clocks, permanent processing.....	13
IL_SysTimeDateToSysTime64.....	180	InSync.....	6
IL_TCPClose.....	279	InTorque.....	6
IL_TCPCloseAsync.....	272	InVelocity.....	6
IL_TCPConnectionAsync.....	259	IP communication	
IL_TCPCyclic.....	311	Byte order.....	257
Connection mode IL_CONN_MODE.....	320		
Connection mode IL_CONN_STATUS.....	321	J	
Error codes.....	315	Jogging variables.....	69
IL_TCPInitial.....	274	Basic rules.....	70
IL_TCPInitialAsync.....	263	Calculate change rate.....	69
IL_TCPRecv.....	277	Components.....	80
IL_TCPRecvAsync.....	268	Introduction and overview.....	69
IL_TCPRecvPacket.....	278		
IL_TCPRecvPacketAsync.....	270	K	
IL_TCPSend.....	276	KIN_OBJECTS.....	24
IL_TCPSendAsync.....	267		
IL_TEMPCONTROL_CONTR_PARAM.....	486	L	
IL_TEMPCONTROL_CONTROL_MODE.....	486	Library	
IL_TEMPCONTROL_CONTROL_TYPE.....	487	IL_VCP_DP.....	497
IL_TEMPCONTROL_EXTENDED_SETTINGS.....	487	RIL_CommonTypes.....	19
IL_TEMPCONTROL_LIMITS.....	488	RIL_HMI_Utility.....	493
IL_TEMPCONTROL_MODEL_PARAM.....	489	RIL_LoopControl.....	327
IL_TempControlType01.....	458	RIL_ModbusTCP.....	195
IL_TempControlType02.....	476	RIL_SocketComm.....	245
IL_TFTPGetFileAsync.....	306	RIL_Utility.....	65
IL_TFTPPutFileAsync.....	309	Linked list.....	184
IL_TimeOfDay.....	180	Little-endian.....	258
IL_TIMEZONE.....	181		
IL_Toggle.....	131	M	
IL_TransportDelayType01.....	378	Mathematical functions	
IL_TransportDelayType02.....	381	IL_InterpolationLinear.....	133
IL_TwoPosCtrlType01.....	489	IL_MaxValue.....	135
IL_UDPClose.....	292	MB_ContinuousAdjustType01 (old).....	72, 75
IL_UDPCloseAsync.....	287	MB_DIFF_EVAL.....	59
IL_UDPCyclic.....	316	MB_EXPECT_WINDOW_MODE.....	60
Connection mode IL_CONN_MODE.....	320	MB_GetModuloType01 (old).....	190
Connection mode IL_CONN_STATUS.....	321	MB_GetModuloType02 (old).....	191
Error codes.....	319	MB_GetModuloType03 (old).....	192
IL_UDPInitial.....	289	MB_GetModuloType04 (old).....	193
IL_UDPInitialAsync.....	282	MB_IncrementalAdjustType01 (alt).....	78
IL_UdpKeys.....	494		
IL_UDPRecvfrom.....	291		
IL_UDPRecvfromAsync.....	286		
IL_UDPSendto.....	290		

MB_INTERPOLATION_MODE.....	60	TCP synchronous functions.....	274
MB_POSITION_EVAL.....	61	Temperature controller.....	458
MB_PROBE_CONFIG.....	63	TFTP.....	257
MB_PROBE_CONFIG, special features of		Time and data calculation.....	174
Sercos-based axes.....	63	Time and date calculation.....	174
MB_PROBE_DATA.....	62	Time-discrete function blocks.....	329
MB_PROBE_INPUT.....	61	TOUCHPROBE_REF.....	25
MB_PROBE_MODULE.....	61	TP_OBJECTS.....	25
MC_CAM_ID	26	Two-position controller.....	489
MC_START_MODE	51	Two-position controller with hysteresis.....	489
MC_STEP_MODE.....	26		
MC_SYNC_DIRECTION	53	U	
ML_InterpolationLinear (old).....	133	UDP.....	250
ML_MaxValue (old).....	135	UDP asynchronous function blocks.....	279
ML_TwoPosCtrlType01 (old).....	489	UDP synchronous functions.....	289
Modbus			
Pro-face GP-Pro EX 2.2.....	202	V	
Siemens WinCC-flexible 2007.....	198	VCP_PBS16_A4096.....	497
Modbus TCP.....	195	VCP_PBS32_A4096.....	498
ModbusTCP client.....	218	VCP_PBS32_A65536.....	499
ModbusTCP server.....	196	VI-Composer.....	498, 499
Modulo value calculation.....	189		
		W	
O		Warnings.....	1
OBJECTS.....	22		
R			
RIL_CommonTypes, data types on the inter-			
face of function blocks.....	19		
RIL_CommonTypes.library.....	19		
RIL_HMI_Utility.library.....	493		
RIL_LoopControl.library.....	327		
RIL_ModbusTCP.library.....	195		
RIL_SocketComm, simplified TCP or UDP			
communication.....	245		
RIL_SocketComm.library.....	245		
RIL_Utility.library.....	65		
S			
Safety instructions.....	1		
Service hotline.....	501		
Shutdown.....	6		
Signal alert symbol.....	2		
Signal generators.....	401		
Signal words.....	2		
Small operator terminal.....	497		
Standard interfaces at function blocks.....	5		
Statistic function blocks.....	429		
Support.....	501		
Symbols.....	3		
Synchronous ICMP functions.....	295		
SYS_TIME_DATE.....	58		
SYS_TIME64.....	57		
T			
TCP.....	246		
TCP asynchronous function blocks.....	258		

Notes

Notes

Bosch Rexroth AG

Electric Drives and Controls

P.O. Box 13 57

97803 Lohr, Germany

Bgm.-Dr.-Nebel-Str. 2

97816 Lohr, Germany

Phone +49 9352 18 0

Fax +49 9352 18 8400

www.boschrexroth.com/electrics



R911398633